Maik Debes

Konzeption und Realisierung eines kontextsensitiven Routingverfahrens

Konzeption und Realisierung eines kontextsensitiven Routingverfahrens

von Maik Debes

Universitätsverlag Ilmenau
2009

Impressum

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Diese Arbeit hat der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität Ilmenau als Dissertation vorgelegen.

Tag der Einreichung:	22. April 2008
1. Gutachter:	Prof. Dr. rer. nat. habil. Jochen Seitz TU Ilmenau
2. Gutachter:	Prof. Dr.-Ing. Günter Schäfer TU Ilmenau
3. Gutachter:	Dr.-Ing. Wolfram Rink Dr. Rink Consult, Frankenhain
Tag der Verteidigung:	26. September 2008

Technische Universität Ilmenau/Universitätsbibliothek
Universitätsverlag Ilmenau
Postfach 10 05 65
98684 Ilmenau
www.tu-ilmenau.de/universitaetsverlag

Herstellung und Auslieferung
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster
www.mv-verlag.de

ISBN 978-3-939473-40-4 (Druckausgabe)
urn:nbn:de:gbv:ilm1-2008000201

Vorwort

Die vorliegende Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am Fachgebiet Kommunikationsnetze der Technischen Universität Ilmenau. Dieses Vorhaben wäre ohne die mir gewährte große Unterstützung nicht realisierbar gewesen. Aus diesem Grund möchte ich mich vor allem bei meinem Betreuer Herrn Prof. Dr. rer. nat. habil. Jochen Seitz bedanken, der stets ein offenes Ohr für Probleme hatte und mir sowohl in organisatorischen als auch inhaltlichen Fragen beratend zur Seite stand. Seine Anregungen und Ratschläge haben mir sehr bei der fachliche Gestaltung der Arbeit geholfen.

Großer Dank gilt auch Herrn Dr.-Ing. Wolfram Rink, der mir mit Tipps und Hinweisen zur Ausgestaltung der schriftlichen Arbeit sehr geholfen hat. Dadurch wurde ich auf einige Problempunkte aufmerksam und konnte diese hoffentlich beseitigen. Seine Visionen und Ideen waren und sind stets eine inspirierende Quelle für mich.

Herrn Prof. Dr.-Ing. Günter Schäfer bin ich sehr dankbar dafür, dass er sich zur Übernahme des Korreferats bereit erklärt hat.

Meinen Kollegen vom Fachgebiet danke ich für die große Unterstützung während meiner gesamten Zeit am Fachgebiet.

Darüber hinaus führten gerade die große Einsatzbereitschaft und Motivation der an dieser Arbeit beteiligten Studenten zu einem erfolgreichen Ergebnis. Ganz besonders möchte ich mich hierbei bei Herrn Marco Wenzel und Herrn Karsten Renhak für die Unterstützung bei der praktischen Realisierung und den vielfältigen Zuarbeiten bedanken. Weiterhin gilt mein Dank Christian Bornträger, Leopold Dombissi, Josa Eberle, Kilian Förster, Heinz Gensicke, Michael Heerwagen, Jana Henniger, Marco Kramer, Alexander Marr, Nadine Niestroy, Sabine Nowak, Guido Paschold, Kay Pein, Christian Potthoff, Enrico Schmidt und Sven Winter.

Abschließend möchte ich mich bei den Menschen bedanken, die mir stets den nötigen Rückhalt im Leben gegeben haben. Meinen Eltern und meiner Schwester Cindy ein großes Dankeschön dafür, dass ihr immer für mich da seid und mich auf jedwede Weise unterstützt. Ebenso gilt dies für meine Freundin Agnieszka, die mich darüber hinaus während der gesamten Promotionszeit „erdulden“ musste. Ihr großes Verständnis gegenüber meiner Arbeit half mir insbesondere bei schwierigen Phasen. Sie wusste es stets, mich zu motivieren und mit Zuversicht in die Zukunft blicken zu lassen. Darüber hinaus war ihre Hilfe bei der Fertigstellung der Arbeit von unschätzbarem Wert.

Kurzzusammenfassung

Das Angebot von Kommunikations- und Informationsdiensten richtet sich einerseits nach den individuellen Bedürfnissen der Nutzer, andererseits aber auch nach den Möglichkeiten der verwendeten Netztechniken. Um einen optimal auf den Nutzer angepassten Dienst bereitstellen zu können, muss dessen Kontext berücksichtigt werden. Ein Dienstzugriff soll dabei möglichst flexibel und unabhängig vom Netz erfolgen. Den mobilen Zugriff auf solche Dienste können Infrastrukturnetze, durch ihre Topologie bedingt, nicht überall bedienen. Mobile Ad-hoc-Netze sind dagegen an keine Infrastruktur gebunden und deshalb sehr flexibel einsetzbar. Durch deren dynamische Topologie ergeben sich jedoch gegenüber den herkömmlichen Netzen auch spezielle Anforderungen an die Routen- und Dienstsuche. Konventionelle Verfahren sind hierbei in der Regel weder nutzbar noch für einen Einsatz in heterogenen Umgebungen geeignet. Gerade mit der fortschreitenden Konvergenz der Netze ist aber eine netzübergreifende Lösung unumgänglich.

Die vorliegende Arbeit befasst sich deshalb mit den Themen Ad-hoc-Netze sowie den hierbei eingesetzten Routingprotokollen und -verfahren. Es werden Möglichkeiten zur Dienstsuche erläutert und gegenübergestellt. Mit den Erkenntnissen daraus wird ein Architekturkonzept entwickelt, das die Dienstsuche sowohl in Ad-hoc- als auch in herkömmlichen Infrastrukturnetzen erlaubt. Die Dienstsuche erfolgt dabei in Verbindung mit einem für diesen Zweck erweiterten Routingverfahren. Warum diese Art einer Dienstsuche auf Netzwerkebene vorteilhaft ist, wird erläutert und begründet. Kernstück der entwickelten Architektur bilden die so genannten Kontextrouter, die eine Dienstsuche unabhängig von der Adresse eines Dienstanbieters unterstützen. Auch der Nutzer benötigt keine Kenntnis über eine solche Adresse. Eine Suche erfolgt lediglich über den Dienst und den zur Verfügung stehenden Informationen über den Kontext des Nutzers. Damit wird ein auf den Nutzer optimal angepasster Dienst ausgewählt. Das Konzept unterstützt die Weiterleitung von Daten zu einem alternativen Server, sofern der ursprüngliche Server ausfällt, und bietet Providern die Möglichkeit, steuernd auf die Dienstsuche und -kommunikation einzuwirken.

Das Verifizieren des vorgestellten Konzeptes erfolgt auf Basis praktischer Realisierungen. Das dazu aufgebaute Demonstratornetzwerk dient für eine Reihe von Tests zum Nachweis der Funktionen und der Leistungsfähigkeit der Architektur. Die daraus resultierenden Ergebnisse beweisen, dass das Konzept den gestellten Anforderungen genügen kann. Abschließend werden in der Arbeit Vorschläge für zukünftige Weiterentwicklungen unterbreitet.

Abstract

Communication and information services both depend on the individual needs of the users and on the possibilities of the used network techniques. To provide an optimal service level adapted to the user, his context must be considered. Thereby, service access should be carried out as flexibly and independently from the network as possible. Infrastructure networks, due to their topology, cannot give mobile access to such services everywhere. Mobile ad hoc networks are not bound to any infrastructure and are therefore very flexibly applicable. As a result of their dynamic topology, there are special requirements to routing and service discovery compared to conventional networks. Conventional methods usually are neither usable in ad hoc networks, nor suitable in heterogeneous environments. Especially because of the progressive convergence of the networks a network-wide solution is inevitable.

This work therefore is concerned with the topics of ad hoc networks as well as routing protocols and procedures which are used for these. Possibilities for service discovery are described and compared. Based on this, an architectural concept is developed, which permits service discovery both in ad hoc networks and in conventional infrastructure networks. Service discovery is combined with a routing protocol especially extended for this purpose. Why this kind of service discovery on network level is advantageous is described and well founded. Principal components of the developed architecture are the so-called context-routers, which support service discovery independently of the address of a service provider. Thus, the user does not need to know such an address. Service discovery is, therefore, depending on the service itself and on the available information about the context of the user. Hence, a service optimally adapted to the user is selected. The concept supports rerouting data messages to an alternative server, if the original server breaks down, and offers the possibility for the provider to control service discovery and communication.

A verification of the presented concept is carried out with practical implementations. The demonstrator network developed for this is used for a set of tests to verify the functions and the efficiency of the architecture. The results prove that the concept can meet the requirements. Finally, suggestions for future advancements conclude this dissertation.

Inhaltsverzeichnis

1. Einleitung

Die vorliegende Dissertation beschreibt eine Architektur zur Dienstsuche in heterogenen Netzwerkumgebungen. Zwar existiert schon eine Reihe von Suchverfahren. Diese berücksichtigen jedoch nur unzureichend den Kontext des Nutzers und die in den letzten Jahren einsetzende Netzkonvergenz. Darauf gehen die folgenden Abschnitte ein und zeigen, welche Mängel bei aktuellen Verfahren zur Dienstsuche diese Arbeit motivierten. Außerdem wird die Zielstellung der Dissertation erläutert. Abschließend wird auf die Gliederung eingegangen und damit die Bedeutung der Kapitel sowie bestehende Zusammenhänge zwischen diesen beleuchtet.

1.1 Motivation

Die Evolution der Netze hängt unmittelbar mit der Evolution der Dienste zusammen. So entwickelte sich das Angebot von Streaming-Diensten im Internet beispielsweise mit Einführung der Netzzugangstechnologie *Digital Subscriber Line* (DSL). Durch diese Evolution bedingt, werden auch neue Verfahren notwendig, die eine effiziente Suche nach Diensten erlauben. Die herkömmliche Diensterbringung in den Netzwerken beruht darauf, dass der Nutzer bzw. dessen Gerät die Adresse des diensterbringenden Servers kennt. Dementsprechend sind die einzelnen Verfahren an das jeweilige Netz und dessen Topologie angepasst. Diese Methode gelangt jedoch bei Ad-hoc-Netzen an ihre Grenzen, da solche Netzwerke eine dynamische Topologie besitzen. Dadurch sind Ad-hoc-Netze zwar äußerst flexibel aber auch sehr komplex in ihrer Funktion. Neue Netzknoten, und damit auch Diensterbringer, können ständig hinzukommen oder das Netzwerk verlassen. Aktuell zur Verfügung stehende Serveradressen können einem Nutzer also erst zum Zeitpunkt der Dienstanfrage bereitgestellt werden.

Hinzu kommt, dass sich in einem Netz mehrere Server befinden können, die den gleichen oder einen ähnlichen Dienst anbieten. In diesem Fall ist es sinnvoll, den Server auszuwählen, der am besten auf den Kontext des Nutzers angepasst ist. Dem kommt auch die aktuelle Entwicklung entgegen, dass Netzteilnehmer dienstorientierter und individueller bedient werden möchten. Der Nutzer selbst interessiert sich dabei nicht für den Server oder dessen Adresse. Er möchte lediglich den für ihn geeignetsten

Dienst nutzen. Deshalb ist eine Anfrage nach einem Service unter Angabe aktuell verfügbarer Kontexttypen sinnvoller, als die Dienstanfrage an eine bestimmte Serveradresse zu richten, bei der nicht gewährleistet werden kann, dass zum Anfragezeitpunkt der Server auch tatsächlich erreichbar ist.

Aktuelle Forschungs- und Entwicklungsarbeiten (z. B. [Eber06, KrDS07]) zeigen, dass herkömmliche Infrastrukturnetze trotz der Entwicklung von Ad-hoc-Netzen nichts an ihrer Attraktivität verloren haben. Werden solche Netzwerke näher untersucht, lassen sich drei aktuelle Trends erkennen. Zum Ersten setzen sich paketvermittelte Übermittlungsverfahren und hierbei speziell die Verwendung des *Internet Protocols* (IP) durch. Zum Zweiten führt gerade die Internettechnologie zum Verschmelzen verschiedener Übertragungstechnologien, sodass mittlerweile Dienste auch in hybriden Netzwerkumgebungen angeboten werden. Zum Dritten führt der Erfolg des Internets zu einer ständigen Entwicklung von neuen Diensten, wobei hier personalisierte bzw. kontextsensitive Dienste immer bedeutender werden. Nicht unwesentlich ist auch der Fakt, dass der Nutzer in solchen Netzwerken mobil sein kann. Für ihn ist also auch hier wieder weniger interessant, über welche Adresse er einen Diensterbringer erreichen kann. Er möchte einfach einen Dienst nutzen, der optimal auf ihn angepasst ist.

Da, wie oben bereits erwähnt, Ad-hoc-Netze eine dynamische Topologie besitzen, ist deren Routing verglichen mit dem in anderen Netzwerken am anspruchsvollsten. Hier erscheint außerdem die Kopplung von Routing und Dienstsuche aus Gründen der Verkehrslast als sehr effektiv. Sofern bei der Dienstsuche auch der Kontext des Nutzers berücksichtigt werden kann, wird dies als kontextsensitives Routing bezeichnet. Um den Ansprüchen in heterogenen Netzwerken genügen zu können, muss somit ein zu entwickelndes Routingprotokoll, das kontextsensitives Routing unterstützt, mindestens den Ansprüchen eines Ad-hoc-Netzes genügen. Erfolgt mit einem solchen Protokoll eine Anfrage nach einem kontextsensitiven Dienst, muss diese bis zu einem geeigneten Server weitergeleitet bzw. geroutet werden. Um dieses Szenario zu unterstützen, ist der Einsatz entsprechender Router erforderlich. Da es sich dabei um eine Erweiterung bestehender Netzwerke handelt, hätte dies auch keine Auswirkungen auf das herkömmliche IP-Routing. Die Idee des kontextsensitiven Routings kann somit eine Möglichkeit der Diensterbringung unabhängig von der Netztopologie darstellen. Da die Dienstsuche auf Routingebene erfolgt, ist ein solches Verfahren sehr effizient, flexibel und zu den anwendungsorientierten Protokollschichten transparent.

Die vorliegenden Arbeit untersucht deshalb Möglichkeiten, eine Architektur für das kontextsensitive Routing zu entwickeln, die den dargestellten Anforderungen genügen kann. Das im Ergebnis vorgeschlagene Konzept basiert dabei auf Untersuchungen zu Ad-hoc-Netzen, zu den dafür geeigneten Routingprotokollen und zu aktuellen Architekturen für eine allgemeine sowie kontextsensitive Diensterbingung bzw. -suche.

1.2 Zielsetzung der Arbeit

Die folgende Arbeit soll untersuchen, inwieweit das kontextsensitive Routing den tatsächlichen Anforderungen einer kontextsensitiven Diensterbringung genügen kann. Ziel dieser Arbeit ist deshalb, eine Architektur zu entwickeln, die eine kontextsensitive Dienstsuche während des Routingprozesses in einem Ad-hoc-Netz unterstützt. Die

Dienstsuche soll dabei mit Rücksicht auf den Nutzerkontext erfolgen. Ein entsprechendes Protokoll ist zu spezifizieren. Das Dienstsuchverfahren muss sowohl in separaten Ad-hoc- und Infrastrukturnetzen als auch in heterogenen Netzwerkumgebungen realisierbar sein. Großer Wert ist hierbei auf die Interoperabilität zu herkömmlichen Netzwerkprotokollen und Verfahren der Dienstsuche zu legen. Den Schwerpunkt bilden hierbei Netzwerke, die das IP in der Version 4 (IPv4 [Post81a]) unterstützen. Daneben sind aber auch die Anforderungen für einen Einsatz unter der Version 6 (IPv6 [DeHi98]) zu berücksichtigen, sodass die Architektur zukünftig möglichst einfach portiert werden kann.

Neben der theoretischen Konzeption der Routingarchitektur soll diese auch simuliert und/oder praktisch umgesetzt werden. Dazu müssen entsprechende Ansätze aufgezeigt und untersucht werden. Für die praktische Realisierung ist ein Demonstrator aufzubauen, der die entwickelte Architektur unterstützt und in ein produktives Netzwerk überführt werden kann. Mit Hilfe dieses Demonstrators können erste Erfahrungen gesammelt und mögliche Schwächen der Architektur aufgezeigt werden. Die dazu durchzuführenden Tests dienen auch zum Nachweis der Funktionen und erlauben erste Aussagen über das Verkehrsverhalten.

Die kontextsensitive Routingarchitektur unterstützt neben der Dienstsuche auch den Transport der zur Erbringung des Dienstes notwendigen Daten. Dagegen bleibt die eigentliche Diensterbringung den Protokollen der höheren Schichten überlassen und ist nicht Bestandteil dieser Arbeit.

1.3 Gliederung der Arbeit

Die Gliederung folgt der Zielsetzung in Abschnitt 1.2.

Das folgende Kapitel 2 widmet sich den Ad-hoc-Netzen. Es wird auf die Eigenschaften solcher Netzwerke eingegangen und gezeigt, welche Unterschiede und Herausforderungen zu herkömmlichen Infrastrukturnetzen bestehen. Damit wird die Basis geschaffen, um eine kontextsensitive Routingarchitektur konzipieren zu können, die auf die Bedürfnisse von Ad-hoc-Netzen angepasst und trotzdem interoperabel zu Infrastrukturnetzen ist. Darüber hinaus werden dort auch Netzwerktechniken beschrieben, die Ad-hoc-Netze unterstützen. Basierend auf diesen Informationen wird dann die am besten geeignete Lösung für die praktische Realisierung innerhalb eines Demonstrators ausgewählt.

Das sich anschließende Kapitel 3 diskutiert aktuelle Routingverfahren für Ad-hoc-Netze. Dazu werden die Anforderungen an ein Routingprotokoll erläutert und die Besonderheiten für Ad-hoc-Netze aufgezeigt. Es wird eine Vielzahl aktueller Routingprotokolle gegenübergestellt und verglichen. Die daraus resultierenden Ergebnisse dienen als Entscheidungshilfe bei der Wahl eines geeigneten Basisprotokolls für das kontextsensitive Routing während der Konzeptionierung der Architektur in Kapitel 6.

In Kapitel 4 werden Begriffe aus dem Bereich der kontextsensitiven Diensterbringung definiert. Diese Definitionen ermöglichen vor allem ein besseres Verständnis für das Thema Kontext. Außerdem wird beschrieben, wie sich der Begriff Dienst in diese Arbeit einordnet und welche Informationen für eine Suche nach kontextsensitiven Diensten relevant sind. Schließlich zeigt das Kapitel auch, warum kein aktuelles Verfahren den Anforderungen einer solchen Dienstsuche gerecht wird.

Kapitel 5 beschreibt, welche Möglichkeiten bestehen, um eine Dienstsuche auf Routingebene durchzuführen. Die verschiedenen Ansätze werden gegenübergestellt und verglichen. Daraus geht das kontextsensitive Routing als beste Lösung hervor. Es überzeugt dabei auch im Vergleich mit den herkömmlichen Verfahren aus Kapitel 4.

Der Ansatz des kontextsensitiven Routings dient nun als Basis für die Entwicklung eines Architekturkonzeptes in Kapitel 6. Die für das Konzept relevanten Lösungsansätze werden vorgestellt und verglichen. Die im Ergebnis entstandene Architektur wird ausführlich beschrieben. Neben der Funktionsweise der beteiligten Netzknoten erfolgt auch eine detaillierte Beschreibung der Kommunikations- und Protokollabläufe.

Kapitel 7 erläutert, warum für die weitere Entwicklung des kontextsensitiven Routings Simulationen als sinnvoll erachtet werden. Außerdem bietet es Lösungsansätze, um das Architekturkonzept in eine vorhandene Simulationsumgebung einbinden zu können.

Vorschläge zur praktischen Realisierung der das Konzept umfassenden Bestandteile erfolgen dann in Kapitel 8. Die hier vorgestellten Implementierungen und deren Funktionen werden beschrieben. Sie bilden die Basis für den im nachfolgenden Kapitel verwendeten Demonstrator.

Kapitel 9 erläutert die Durchführung verschiedener Tests und Messungen mit Hilfe eines dafür errichteten Demonstratornetzwerkes. Die Ergebnisse der Untersuchungen dienten dazu, die Funktionsfähigkeit der Architektur nachweisen und erste Aussagen zur Leistungsfähigkeit treffen zu können.

Kapitel 10 erläutert Vorschläge zu zukünftigen Forschungs- und Entwicklungsarbeiten, um später einen produktiven Einsatz des kontextsensitiven Routings zu ermöglichen.

Abschließend fasst Kapitel 11 die Arbeit sowie deren Ergebnisse zusammen und bewertet diese.

Es wird darauf hingewiesen, dass alle im Text enthaltenen englischen Begriffe beibehalten und durch kursive Schrift gekennzeichnet wurden. Die Kennzeichnung entfällt jedoch bei solchen Begriffen, die sich bereits in den deutschen Sprachgebrauch eingefügt haben (z. B. Internet).

2. Ad-hoc-Netze

Soll ein kontextsensitives Routingverfahren entwickelt werden, das in einer heterogenen Netzwerkumgebung arbeitet, d. h. verschiedene Netzarten unterstützt, muss es den Anforderungen des komplexesten Netzes genügen. Diesbezüglich sind derzeit mobile Ad-hoc-Netze am anspruchsvollsten. Woraus sich diese Aussage ergibt und welche Eigenschaften diese Netze besitzen, zeigt das folgende Kapitel auf. Es führt dazu in den Aufbau und die Funktionsweise von Ad-hoc-Netzen ein. Außerdem wird gezeigt, welche aktuellen Techniken diese Netze unterstützen und welche Herausforderungen sich gerade für die Netzzugangsschicht ergeben. Das Kapitel bildet damit auch eine Grundlage für die im anschließenden Kapitel 3 geführte Diskussion über das Routing in Ad-hoc-Netzen.

2.1 Einführung

Drahtlose mobile Netze können in Infrastruktur- und Ad-hoc-Netze gegliedert werden. Bei Infrastrukturnetzen erfolgt die Kommunikation zwischen den Teilnehmern immer über einen Zugangspunkt, dem so genannten Access Point bzw. der Basisstation. Abbildung 2.1 zeigt ein Beispiel für ein solches Netz. Die Basisstationen sind hierbei stationär und direkt oder indirekt miteinander verbunden. Sie können die Daten ihrer mobilen Teilnehmer untereinander austauschen und somit große Netzwerke bilden. Bei Infrastrukturnetzen ist immer nur der Teilnehmer mobil. Außerdem wird für die Kommunikation mindestens ein Hop[1] bis zum (mobilen) Nachbarn und maximal ein Hop bis zum stationären Netz benötigt. Daraus folgt auch, dass die Teilnehmer niemals direkt miteinander kommunizieren. Geräte, die sich außerhalb der Funkreichweite der Basisstationen befinden, sind damit von einer Kommunikation ausgeschlossen – selbst dann, wenn sich benachbarte Netzknoten in unmittelbarer Nähe befinden und diese theoretisch als Relay-Stationen verwendet werden könnten. Allerdings sind die Endgeräte in Infrastrukturnetzen verhältnismäßig einfach gehalten, da sie keine Netzwerkmanagementfunktionen erbringen müssen. Voraussetzung für den Aufbau von Infrastrukturnetzen ist eine vorausgehende Planung. So muss unter anderem festgelegt werden, wo und in welcher Größenordnung Basisstationen

[1] Ein Hop entspricht einem Sprung aus einer vermittelnden Instanz.

aufgebaut werden sollen, was wiederum von der zu erzielenden Abdeckung abhängt. Bekannte Beispiele hierfür sind WLANs (*Wireless Local Area Networks*), die im Infrastruktur-Modus arbeiten, aber auch die aktuellen öffentlichen Mobilfunknetze. So arbeiten unter anderem GSM (*Global System for Mobile Communications*) – incl. dessen Erweiterungen HSCSD (*High Speed Circuit Switched Data*), GPRS (*General Packet Radio Service*), EDGE (*Enhanced Data Rates for GSM Evolution*) – und UMTS (*Universal Mobile Telecommunication Service)* als Infrastrukturnetze.

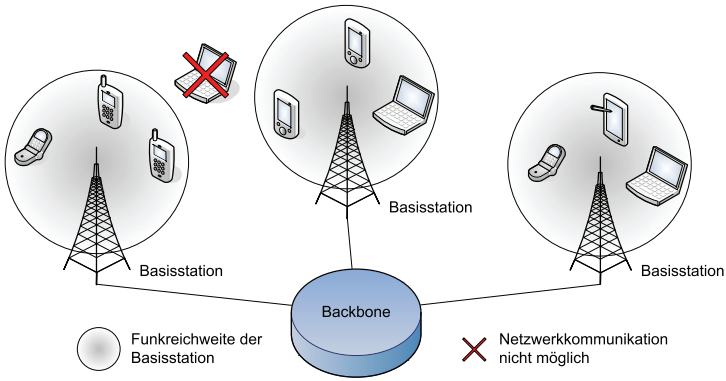


Abbildung 2.1: Drahtloses Infrastrukturnetz

Bei der Planung von Infrastrukurnetzen müssen auch bereitzustellende Dienste wie Namensauflösung, Authentifizierung o. ä. berücksichtigt werden. Die Verwaltung und Administration erfolgt in der Regel zentral. Diese Netze haben aber auch einen entscheidenden Nachteil. Sie sind sehr anfällig gegenüber Natur-, Umwelt- oder von Menschen verursachten Katastrophen. Ist die Infrastruktur zerstört, wird viel Zeit benötigt, um ein neues Netz aufzubauen bzw. zu rekonstruieren. Darüber hinaus können sich Infrastrukturnetze nicht dynamisch an Lastveränderungen anpassen. Einmal dimensioniert arbeiten sie für eine bestimmte Teilnehmeranzahl optimal. In Ausnahmesituationen (z. B. Silvester) ist dann jedoch mit Kommunikationsschwierigkeiten zu rechnen. Ein weiterer großer Nachteil besteht darin, dass es aus ökonomischen Gesichtspunkten unvorteilhaft ist, solche Netze in dünn besiedelten Gebieten aufzubauen. So müssen bestimmte Gruppen, für die solche Gebiete interessant sind, wie beispielsweise Forscher, Wissenschaftler aber auch das Militär, nach anderen Kommunikationsmöglichkeiten suchen.

Eine solche Möglichkeit stellen die aktuell in Erforschung und Erprobung stehenden Ad-hoc-Netze oder mobilen Ad-hoc-Netze (MANET – *Mobile Ad Hoc Network*) dar. Im Folgenden werden Ad-hoc-Netz und MANET synonym behandelt. Solche Netze bieten einen völlig anderen Ansatz. Eine gute Definition liefert [ChMi01]. Danach sind mobile Ad-hoc-Netze:

Definition 2.1 (Ad-hoc-Netze) *„... a collection of wireless nodes, all of which may be mobile, dynamically create a wireless network among themselves without using any such infrastructure or administrative support.“*

Bei dieser Art von Netz fehlt also jegliche Infrastruktur. Eine Kommunikation zwischen zwei Knoten muss trotz dynamischer Topologieänderungen unterstützt werden. Außerdem können alle an einer Kommunikation beteiligten Knoten (hier im Sinne von Netzknoten bzw. Teilnehmerendgeräten) mobil sein. Die Knoten kommunizieren dabei mit mindestens einem ihrer direkten Nachbarn. Liegen zwei Kommunikationspartner außerhalb ihrer Funkreichweite, können andere dazwischen liegende Knoten als Relay-Stationen (so genannte Zwischenknoten) verwendet werden. Somit muss jedes an einem Ad-hoc-Netz beteiligte Gerät auch als Router arbeiten können. Dieses Szenario ist in Abbildung 2.2 dargestellt. Knoten A kann dort mit allen (direkten) Nachbarn kommunizieren. Knoten E befindet sich jedoch außerhalb der Funkreichweite von A. Die Knoten B und D routen die Daten deshalb an den Knoten E weiter.

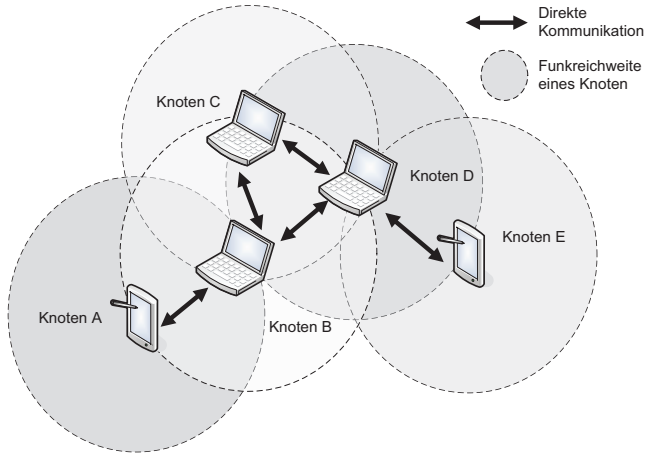


Abbildung 2.2: Mobiles Ad-hoc-Netz

Mit der dargestellten Funktionsweise ergeben sich Anwendungsgebiete, die mit herkömmlichen drahtlosen Infrastrukturnetzen nicht erschlossen werden konnten. Beispielhaft sollen hier zwei Einsatzgebiete kurz beschrieben werden.

- Schule/Universität: Lehrer und Schüler können während des Unterrichts mit ihren Laptops o. ä. spontan lokale Netzwerke aufbauen. Dadurch kann an Aufgabenstellungen gemeinsam und interaktiv gearbeitet werden. Das Netz ist von der örtlich zur Verfügung stehenden Netztechnik unabhängig. Die räumliche Verlagerung von Unterrichtsstunden ist damit unproblematisch. Ein solches Szenario wird beispielsweise von den Geräten des Ausbildungsprojektes „One Laptop per Child“ [OLPC08] unterstützt. Kindern auf der ganzen Welt soll somit der Zugang zu Computern und gemeinsames Lernen ermöglicht werden. Deshalb wurden kostengünstige Laptops entwickelt, die auch in Gebieten mit gering ausgebauter bzw. keiner Telekommunikationsinfrastruktur eingesetzt werden können. Zum Datenaustausch unterstützen diese eine Kommunikation via Ad-hoc-Netz.

- Telematikanwendungen im Verkehr: Beispielsweise könnte, wie in Abbildung 2.3 dargestellt, Netzwerktechnik in Kraftfahrzeugen implementiert werden. Dadurch wären die Verkehrsteilnehmer in der Lage, gesammelte aktuelle Informationen an benachbarte Teilnehmer zu senden (z. B. Stauinformationen). Jedes Kraftfahrzeug wird auch als Router genutzt, sodass eine Informationsweiterleitung über größere Strecken erfolgen kann. Staus können so beispielsweise früh erkannt und die persönliche Streckenplanung entsprechend geändert werden.

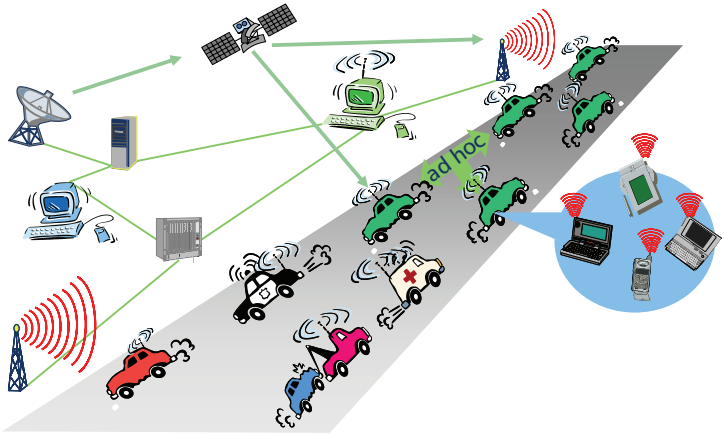


Abbildung 2.3: Vernetzung von Verkehrsteilnehmern [Schi03]

In den genannten Beispielen kommunizieren die Teilnehmer über ein drahtloses aber vermaschtes Netz. Bevor hier eine Kommunikation erfolgen kann, muss ein entsprechender Weg zwischen den jeweiligen Kommunikationspartnern gefunden werden. Dabei wird abhängig vom verwendeten Wegewahlverfahren ggf. auch eine Lastverteilung berücksichtigt.

Eine weitere Strategie verfolgen die sogenannten opportunistischen Netze (*Opportunistic Networks* [PePC06]). Diese arbeiten verzögerungstolerant und ohne herkömmliche Routenfindung zwischen Quelle und Ziel. Eine Nachricht wird gesendet und sucht sich von Knoten zu Knoten den besten Weg, um das Ziel zu erreichen. Hierbei kann es durchaus passieren, dass die Nachricht auf dem aktuellen Knoten zwischengespeichert wird, bis eine geeignete Weiterleitung zum Ziel gefunden wird. Opportunistische Netze sind zwar gegenwärtig Mittelpunkt vielfältiger aktueller Forschungsaktivitäten, werden aber in den folgenden Betrachtungen nicht gesondert behandelt, da dieses Thema durch die Ad-hoc-Netze sowie durch Aussagen zur Unterstützung hybrider Netze (z. B. Abschnitt 6.2.4) ausreichend berücksichtigt wird.

Wie in diesem Abschnitt dargestellt, müssen Ad-hoc-Netze ein Höchstmaß an Flexibilität unterstützen. Welche Eigenschaften sich daraus für solche Netze ergeben, zeigt der folgende Abschnitt.

2.2 Eigenschaften von Ad-hoc-Netzen

Zwei große Vorteile, die in engem Zusammenhang mit dem infrastrukturlosen Aufbau von Ad-hoc-Netzen stehen, sind, dass diese Netze keiner vorausgehenden Planung bedürfen und über ein dezentrales Netzmanagement organisiert werden. Sie sind relativ unanfällig gegenüber äußeren Einflüssen und damit auch hervorragend für den schnellen Einsatz in Katastrophengebieten geeignet.

Das dezentrale Netzmanagement kann mit drei Merkmalen zusammengefasst werden – „*self-creating*“, „*self-organizing*“ und „*self-administering*“. D.h., ein Netz kann spontan aufgebaut werden, ohne dass der Nutzer administrativ eingreifen muss. Die Knoten müssen also auch in der Lage sein, sich selbständig in dieses Netz zu integrieren. Erst wenn diese Voraussetzungen erfüllt sind, wird es als Ad-hoc-Netz bezeichnet. Somit muss jeder Netzknoten ein gleiches Mindestmaß an Funktionalität unterstützen. Allerdings erhöht dies die Komplexität der Knoten, da jeder einzelne die gesamte Netzintelligenz besitzt. Im Gegensatz dazu befindet sich die Netzintelligenz bei Infrastrukturnetzen hauptsächlich im Netzzugangspunkt und dem als Backbone dienenden Festnetz.

Die Topologie des Netzes ermöglicht im einfachsten Fall eine direkte Kommunikation zwischen zwei Endgeräten. Aufgrund der verwendeten Funkstandards (z. B. WLAN) besitzen die an einem Ad-hoc-Netz beteiligten Geräte eine entsprechend begrenzte Reichweite. Deshalb muss jedes Gerät in der Lage sein, als Router arbeiten zu können, um ggf. als Weiterleitungsstation zu dienen. Prinzipiell sind also Multihop-Verbindungen über die mobilen Knoten möglich. Die Wegewahl in einem solchen Netz erfolgt mit Hilfe spezieller Routingverfahren. Auf die dort zu berücksichtigenden Probleme und die Verfahren selbst wird in den Kapiteln 3 und 5 eingegangen.

Da drahtlose Ad-hoc-Netze Funktechniken verwenden, müssen deren besondere Eigenschaften berücksichtigt werden. Funknetze nutzen beispielsweise abhängig vom Verfahren entsprechende Frequenzbereiche. Die Anzahl der nutzbaren Kanäle unterscheidet sich dann je nach Verfahren. Daraus folgt, dass die Zahl gleichzeitig kommunizierender Knoten innerhalb eines Gebietes begrenzt ist bzw. sich direkt auf die nutzbare Bitrate auswirkt. Des Weiteren können Störungen auf den Funkstrecken auftreten, die schwer vorhersagbar bzw. berechenbar sind. Diese Eigenschaften betreffen aber auch drahtlose Infrastrukturnetze.

Neben der großen Anzahl von Vorteilen der Ad-hoc-Netze bleibt aber auch noch eine Vielzahl von Fragen offen. So werden beispielsweise gegenwärtig Forschungen dazu durchgeführt, inwieweit Ad-hoc-Netze mit anderen Netzwerken zusammenarbeiten (Interoperabilität) oder wie Dienstgüte bzw. Quality of Service (QoS) bereitgestellt werden können. Die klassischen Ansätze, wie sie aus dem Festnetz bekannt sind, versagen hier häufig. Ein großes Problem stellt weiterhin die Energieversorgung der mobilen Knoten dar. Gegenwärtig werden diese mit herkömmlichen Batterien oder Akkumulatoren gespeist. Betriebsdauer, Reichweite und Bewegungsfreiheit sind dadurch aber stark begrenzt. Die Zuverlässigkeit der Kommunikation zwischen zwei Knoten sinkt darüber hinaus, weil auch ein sich dazwischen befindender Knoten, der als Router verwendet wird, wegen Energiemangel ausfallen kann und deswegen die Kommunikation unterbrochen wird. Weiterhin ist auch ungeklärt, wie die Sicherheit in einem solchen Netz gewährleistet werden kann. Z. B. muss berücksichtigt werden, dass prinzipiell jeder Nutzer Daten anderer Nutzer weiterleiten und seine

eigene Kommunikation mit Hilfe der Geräte anderer Nutzer erfolgen kann. Es ist also zum einen sicherzustellen, dass eine Kommunikation nicht von Dritten abgehört werden kann. Zum anderen dürfen Nutzer, die einen Knoten zur Weiterleitung von Daten verwenden, darauf keinen Schaden (z. B. Einbruch, Systemabsturz usw.) verursachen können. Diese Problematik wird beispielsweise in [Gren04] diskutiert. Dort werden auch einige Vorschläge erörtert, um die Sicherheit in Ad-hoc-Netzen zu erhöhen. Darüber hinaus ist bis jetzt auch der Zugang zu den verschiedensten Diensten ungeklärt. Die Schwierigkeit beim Zugriff auf einen Dienst besteht nämlich darin, dass die IP-Adresse des zugehörigen Servers einerseits nicht bekannt ist, andererseits auch nicht gewährleistet werden kann, dass dieser Server überhaupt im Netz existiert. Das betrifft beispielsweise Webdienste, Fileserver, Server zur Namensauflösung usw. Zwar gibt es hierfür erste Ansätze (siehe [ToGM04, LiLa05]), die sich allerdings bis jetzt nicht durchsetzen konnten. Übergangsweise wird auch versucht, mit bekannten Systemen aus den Infrastruktur- bzw. Festnetzen zu arbeiten. Diese Verfahren stoßen jedoch im Bereich der Ad-hoc-Netze an ihre Grenzen.

An dieser Stelle muss auch darauf hingewiesen werden, dass es von der Lösung der zuletzt genannten Probleme abhängen wird, inwieweit sich Ad-hoc-Netze allgemein durchsetzen können. Ein Nutzer wird kein unsicheres unzuverlässiges Netz für seine Geschäfte, zum Abruf von Diensten oder einfach zur Kommunikation verwenden. Sicherheit (*Security*) und Vertrauen (*Trust*) sind für eine allgemeine Akzeptanz bei potentiellen Nutzern unerlässlich. Sehr interessant ist an dieser Stelle auch die Frage, wie offen ein System sein darf bzw. sein muss. Darf beispielsweise ein Nutzer dazu gezwungen werden, dass er sein in das Netz involvierte Gerät als Router zur Verfügung stellt oder kann er das Netz zwar benutzen, braucht aber sein Gerät zur Weiterleitung von Daten nicht zur Verfügung zu stellen? Diese Frage kann wohl letztlich nur über Kosten- und Tarifmodelle geregelt werden. Einen ersten Ansatz dafür beschreibt beispielsweise [Klei06]. Bei dem dort vorgestellten und bereits realisierten Konzept teilen sich mehrere registrierte Nutzer einen WLAN-Access-Point für den Zugriff auf das Internet. Dieser Access Point wird von einem Nutzer privat (z. B. via DSL) bereitgestellt. Ein Tarifmodell sieht dabei vor, dass der Besitzer des zur Verfügung gestellten Access Points selbst Zugriff auf andere zur Gemeinschaft gehörende Access Points erhält. Bei einem anderen Tarif erhält er dagegen einen Unkostenbeitrag für die Nutzung. Diese einfachen Varianten lassen sich auch auf Ad-hoc-Netze abbilden. Sicherlich gibt es hier die vielfältigsten Möglichkeit für eine Tarifierung, was aber nicht weiter Bestandteil dieser Arbeit sein soll.

Anhand der hier dargelegten Fakten zu den Eigenschaften und zum Einsatzzweck der Infrastruktur- und Ad-hoc-Netze ist relativ deutlich abzusehen, dass Infrastrukturnetze und Ad-hoc-Netze zukünftig eher weniger miteinander konkurrieren als sich vielmehr ergänzen und somit ihren Anteil zum so genannten „*pervasive and ubiquitous Computing*“ beitragen werden. Aktuelle universitäre Forschungen wie beispielsweise [Eber06] sind ein Beleg dafür. Auch große Provider bestätigen diesen Trend, zumal beispielsweise Festnetzanschlüsse nach deren Aussage [Koss06] auch in zehn Jahren immer noch einen Geschwindigkeitsvorteil von Faktor 10 gegenüber den Funktechniken besitzen werden.

Zum Schluss dieses Abschnittes soll ein kleines Beispiel noch einmal die Komplexität von Ad-hoc-Netzen verdeutlichen. In Abbildung 2.4 bewegen sich die Netzknoten A und B geradlinig gleichförmig in entgegengesetzter Richtung. Ab dem Zeitpunkt t_0

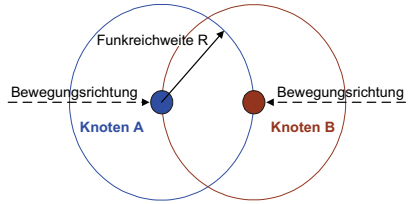


Abbildung 2.4: Bewegung zweier kommunizierender Netzknoten

befinden sich beide Knoten in Funkreichweite zueinander und können direkt kommunizieren. Die Knoten bewegen sich aneinander vorbei und entfernen sich dann wieder. Zum Zeitpunkt t_1 bricht dann die Kommunikation ab. Für dieses Szenario zeigt Tabelle 2.1 Beispiele für die maximale Zeitdauer einer solchen Kommunikation in Abhängigkeit verschiedener Funkreichweiten und Geschwindigkeiten. Die Zeit t, die dafür zur Verfügung steht, ist direkt proportional zur Reichweite R und indirekt proportional zur Geschwindigkeit v eines Knotens. Bewegen sich beide Knoten gleichzeitig und gleich schnell, führt dies zur Verdoppelung der Geschwindigkeit. Die Gesamtstrecke der Kommunikation, die während der Kommunikation zurückgelegt wird, ist gleich der zweifachen Reichweite. Damit ergibt sich bei kreisförmigen Richtcharakteristiken der Antennen folgende bekannte Formel:

$$t = \frac{2R}{2v} = \frac{R}{v} \qquad \text{mit} \qquad t = t_1 - t_0 \tag{2.1}$$

v in km/h \ R in m	10	20	50	100	200
5	7,2 s	14,4 s	36 s	72 s	144 s
10	3,6 s	7,2 s	18 s	36 s	72 s
20	1,8 s	3,6 s	9 s	18 s	36 s
30	1,2 s	2,4 s	6 s	12 s	24 s

Tabelle 2.1: Kommunikationsdauer zwischen zwei mobilen Knoten

Damit kann beispielsweise eine Kommunikation bei einer Geschwindigkeit von 5 km/h pro Knoten maximal 72 s aufrecht erhalten werden, wenn die Funkreichweite für jeden Knoten 100 m beträgt. Danach müssen alternative Routen über Zwischenknoten gesucht und die Routingtabellen aktualisiert werden. Davon sind auch die beteiligten Zwischenknoten betroffen. Außerdem wiederholt sich diese Prozedur nach max. 36 s wieder, sofern die Zwischenknoten nicht mobil sind und auf der Bewegungsgeraden liegen. Dies ist jedoch sehr unwahrscheinlich, sodass mit weitaus geringeren Zeiten gerechnet werden kann. Selbst für dieses einfache Beispiel würde die Prozedur bei einer Kommunikationsdauer von nur 10 Minuten 16 mal wiederholt werden, wenn auch die Wegesuche bei Initiierung der Kommunikation in die Berechnung mit einfließt. Zu beachten ist, dass hier noch nicht die Mobilität der Zwischenknoten, äußere Einflüsse auf die Reichweite, die Abhängigkeit zwischen Empfangsleistung, Reicheite sowie Bitrate usw. berücksichtigt sind. Diese tragen zu einer weiteren Verringerung von t und damit zu einer Erhöhung der Komplexität bei.

2.3 Übertragungstechniken

Ad-hoc-Netze oder MANETs umfassen typischerweise die Schichten 1 bis 3 des ISO/OSI-Referenzmodells[2] bzw. die unteren beiden Schichten (Rechner-Netzanschluss und Internetschicht) des TCP/IP-Modells[3]. Den Netzzugang bilden dabei herkömmliche drahtlose Netzwerke, deren Spezifikation den Ad-hoc-Modus unterstützen. Diesbezüglich werden in der Literatur als Beispiele am häufigsten HIPERLAN (*High Performance Radio Local Area Network*), WLAN und Bluetooth aufgezählt. Diese Spezifikationen beinhalten lediglich die Funktionen der ersten und zweiten Schicht des ISO/OSI-Referenzmodells. Die Adressierung erfolgt dort typischerweise über *Media Access Control*-Adressen (MAC-Adressen). Funktionen der Vermittlungsschicht sind nicht implementiert und müssen für ein Ad-hoc-Netz, wie es in den vorherigen Abschnitten beschrieben wurde, noch aufgesetzt werden. Eine Gegenüberstellung von Techniken für Infrastruktur- und Ad-hoc-Netze zeigt Abbildung 2.5.

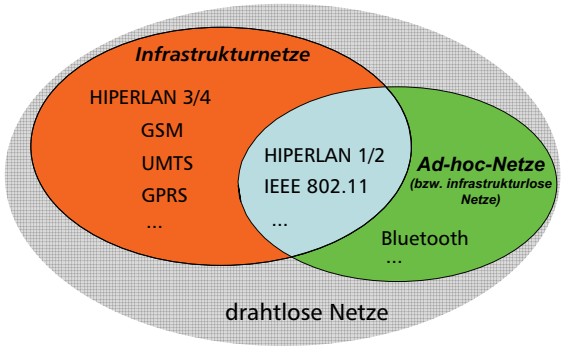


Abbildung 2.5: Netztechniken für Infrastruktur- und Ad-hoc-Netze

Die verschiedenen HIPERLAN-Varianten wurden vom *European Telecommunications Standards Institute* (ETSI) entwickelt und standardisiert. Ad-hoc-Netze können jedoch nur mit HIPERLAN 1/2 realisiert werden. Diese arbeiten im lizenzfreien ISM[4]-Band von 5 GHz. Sie unterstützen Entfernungen von bis zu 50 m innerhalb von Gebäuden sowie bis zu 150 m im freien Feld. HIPERLAN 1 erreicht dabei Übertragungsraten von 23,5 MBit/s. Bei HIPERLAN 2 sind es dagegen 54 MBit/s. Die Sendeleistung beträgt bei beiden Varianten max. 1 W. Leider wurde während des Entwicklungsprozesses versäumt, solche Firmen und Hersteller mit einzubeziehen, die auch entsprechende Produkte bzw. Telekommunikationshardware produzieren können. So besteht seitens der Hersteller bis heute kaum ein Interesse daran, Geräte anzufertigen, die diesen Standard unterstützen. Diese konzentrieren sich vielmehr auf Techniken wie WLAN und Bluetooth.

WLAN (IEEE[5] 802.11) unterstützt neben dem Infrastruktur- auch den Ad-hoc-Modus. In diesem Modus können Knoten, die sich in Funkreichweite befinden, di-

[2]ISO *International Organization for Standardization* (von gr. „isos“)
OSI *Open Systems Interconnection*
[3]TCP *Transmission Control Protocol*
[4]ISM *Industrial, Scientific, and Medical*
[5]IEEE *Institute of Electrical and Electronics Engineers*

rekt miteinander kommunizieren. WLAN nutzt dabei ebenfalls die Frequenzen aus den lizenzfreien ISM-Bändern (2,4 GHz bzw. 5 GHz). Es werden Bruttobitraten von 11 Mbit/s (IEEE 802.11b) und 54 Mbit/s (IEEE 802.11a/g) unterstützt. Wie [Hess05] belegt, ist damit durchaus auch eine Übertragung multimedialer Daten möglich. Im neuesten Standard sollen sogar bis zu 600 Mbit/s (802.11n [W11n07]) erreicht werden. Die in Europa maximal zulässige Sendeleistung beträgt 100 mW im 2,4-GHz- und 1 W im 5-GHz-Bereich. Damit können Reichweiten von mehreren 100 Metern erreicht werden, was jedoch sehr stark von den äußeren Bedingungen abhängt. Hinzu kommt, dass die Bitrate mit größer werdender Entfernung abnimmt. Aufgrund der verwendeten Zugriffsverfahren tritt derselbe Effekt auch mit Erhöhung der Anzahl der Teilnehmer auf. Tabelle 2.2 zeigt in einer kurzen Übersicht die derzeit aktuellen Varianten des WLAN-Standards 802.11. Die Spezifikation für den Standard IEEE 802.11n ist noch nicht verabschiedet, deshalb muss die Tabelle 2.2 vorerst unvollständig bleiben.

802.11	a	b	b+	g
Frequenzband (Europa)	5 GHz	2,4 GHz	2,4 GHz	2,4 GHz
Kanalbandbreite	20 MHz	22 MHz	22 MHz	22 MHz
Anzahl der Kanäle	19	13	13	13
Übertragungsrate (brutto)	54 Mbit/s	11 Mbit/s	22 Mbit/s	54 Mbit/s
max. Sendeleistung	1 W	100 mW	100 mW	100 mW

Tabelle 2.2: Vergleich verschiedener WLAN-Standards [SDHT07]

Bluetooth (IEEE 802.15.1) ist ein Funkstandard, der aus einem Zusammenschluss verschiedener Firmen heraus entwickelt wurde. Zu diesen Firmen zählen beispielsweise Ericsson, Nokia, IBM und Toshiba, die 1998 die Bluetooth-SIG[6] gründeten. Ursprünglich wurde diese Funktechnik konzipiert, um Peripheriegeräte drahtlos und möglichst einfach mit einem PC zu verbinden. Mittlerweile sind die Anwendungen aber weitaus vielfältiger geworden, sodass Bluetooth in allen Bereichen der Kommunikationstechnik wiederzufinden ist. Die Technik arbeitet wie WLAN und HIPERLAN 1 im ISM-Band von 2,4 GHz. Da dieses Band in verschiedenen Ländern unterschiedlich breit ist, werden dort entweder 23 oder 79 Trägerfrequenzen unterstützt. Die Kanalbandbreite beträgt 1 MHz. Bis zur Version 1.2 wird eine Bruttobitrate von 1 Mbit/s unterstützt. Version 2.0 unterstützt bis zu 3 MBit/s [Blue04]. Gegenwärtig wird an der Version 3 gearbeitet, die noch einmal eine Bitratenerhöhung liefern wird. Um die Übertragung mit Bluetooth störungssicherer zu gestalten, werden Frequenzsprungverfahren eingesetzt. Mit den derzeit spezifizierten Leistungsklassen von 1 mW, 10 mW und 100 mW sind Reichweiten von bis zu 100 m möglich. Im Outdoorbereich können durchaus auch größere Entfernungen erreicht werden [Gren02].

Aufgrund des ursprünglich vorgesehenen Einsatzfeldes unterscheidet sich die Netzphilosophie bei Bluetooth erheblich von der bei WLAN. Bluetooth wurde für so genannte Ad-hoc-Piconetze konzipiert. Ad-hoc deshalb, weil der Nutzer möglichst wenig Know-how braucht, um ein Gerät in so ein Netz einzubinden. Die Netze sollen

[6]SIG *Special Interest Group*

sich auch hier, wie in Abschnitt 2.2 beschrieben, möglichst selbständig und ohne Eingriff des Nutzers verbinden. Deshalb werden standardisierte Profile verwendet. Wollen zwei Geräte über eine Anwendung miteinander kommunizieren, müssen beide das gleiche und vor allen Dingen das zur Anwendung gehörende Profil verwenden. Die Profile werden ständig erweitert und den aktuellen Bedürfnissen sowie dem Stand der Technik angepasst. Soll z. B. ein IP-Netzwerk unter Nutzung von Bluetooth realisiert werden, könnte das Profil *Personal Area Network* (PAN) verwendet werden. Hier wird das Bluetooth-spezifische *Bluetooth Network Encapsulation Protocol* (BNEP) eingesetzt. Dieses bildet dann die Basis für die Protokolle der IP-Suite. Detaillierte Beschreibungen zu den Profilen bzw. zum Protokollstack sind beispielsweise in [Hoff04], [Förs05] und [Blue07] zu finden.

In einem Piconetz wechseln beim Frequenzsprung alle beteiligten Geräte auf dieselbe neue Frequenz. Allerdings ist die Anzahl der Geräte innerhalb eines Netzes begrenzt. Es können zwar bis zu 256 Geräte zu einem Piconetz gehören, jedoch maximal acht Endgeräte gleichzeitig miteinander kommunizieren. Ein Gerät muss dabei die Rolle des so genannten Masters übernehmen, alle anderen Geräte stellen die Slaves dar. Der Master steuert die Kommunikation im Netz. Außerdem wird jegliche Kommunikation über ihn geführt. Die Slaves können also nicht direkt miteinander kommunizieren. Solche Einschränkungen würden jedoch die Nutzbarkeit des Netzes sehr eingrenzen. Sollen deshalb mehr als acht Geräte miteinander kommunizieren, helfen so genannte Scatternetze weiter. Wie in Abbildung 2.6 dargestellt, wird darunter die Kopplung mehrerer Piconetze verstanden. Ein Gerät ist hierbei gleichzeitig Mitglied in beiden zu verbindenden Netzen. Das Gerät kann in diesen Netzen jeweils die Rolle eines Slaves übernehmen bzw. in einem Netz ein Slave und in dem anderen Netz ein Master sein, niemals aber in beiden Netzen als Master arbeiten.

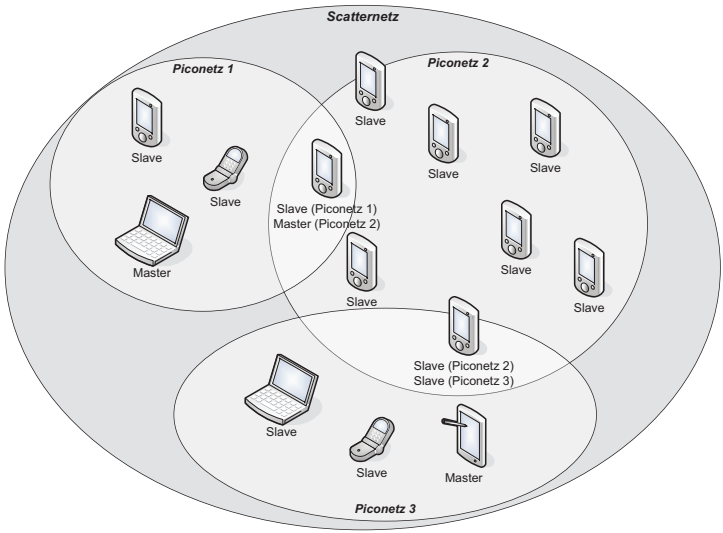


Abbildung 2.6: Pico- und Scatternetz [SDHT07]

Da sowohl WLAN als auch Bluetooth im 2,4-GHz-ISM-Band arbeiten, bleiben gegenseitige Störungen beim gleichzeitigen Betrieb beider Standards nicht aus. Davon ist Bluetooth jedoch weniger betroffen als WLAN. Dies wurde beispielsweise durch Messungen in [HiPa04] nachgewiesen. Bluetooth arbeitet mit schmalbandigen Kanälen und einem Frequenzsprungverfahren. Hier ist der Einfluss von Störungen deshalb geringer. WLAN nutzt dagegen einen fest gewählten Kanal, der zudem breitbandiger ist. Somit werden hier zum Zeitpunkt der Störung wesentlich größere Pakete beschädigt als bei Bluetooth. Diesen Umstand versucht Bluetooth in der Version 1.2 unter Nutzung des AFH (*Adaptive Frequency Hopping*) zu beseitigen. Dieses Verfahren überprüft, welche Frequenzen bzw. Kanäle schon belegt sind und lässt diese beim Sprung zu einer anderen Trägerfrequenz außen vor. Dieser Mechanismus soll dafür sorgen, dass WLAN-Geräte durch Bluetooth weniger gestört werden.

Bei Funkverfahren, die den Medienzugriff im Gegensatz zu Bluetooth nicht zentral steuern, kann aber noch eine weitere Problematik auftreten. Über den Medienzugriff wird geregelt, wann welcher Teilnehmer einen Kanal benutzen darf. Bedingt durch die oben beschriebene Reichweitencharakteristik von WLAN können hier die herkömmlichen Zugriffsverfahren aus den drahtgebundenen Netzen nicht verwendet werden, da das so genannte Hidden-Node- und das Exposed-Node-Problem zu berücksichtigen sind.

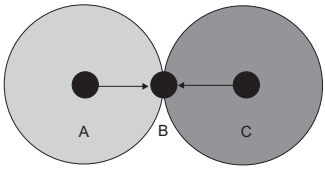


Abbildung 2.7: Hidden-Node-Problem [SDHT07]

Das Hidden-Node-Problem ist in Abbildung 2.7 dargestellt. Es beschreibt den Einfluss entfernter und für den aktuellen Sender verborgener Netzknoten auf die Kommunikation zwischen ihm und seinen Nachbarknoten. In Abbildung 2.7 möchte Knoten A an Knoten B Daten senden. Gleichzeitig sendet Knoten C auf dem gleichen Kanal an Knoten B. Damit empfängt Knoten B nur gestörte Datenpakete. Die Sender (Knoten A und C) selber merken davon aber nichts und versuchen weiterhin auf dem für sie scheinbar freien Kanal zu senden

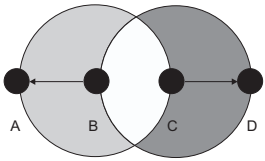


Abbildung 2.8: Exposed-Node-Problem [SDHT07]

Das Exposed-Node-Problem beschreibt den Einfluss exponierter Netzknoten, d. h. den Fall eines scheinbar belegten aber tatsächlich nutzbaren Kanals, was zu einer ineffizienten Ausnutzung des Frequenzspektrums führt und somit freie Kapazitäten ungenutzt lässt. Abbildung 2.8 verdeutlicht dieses Phänomen noch einmal. Knoten B möchte Daten zu Knoten A und Knoten C Daten zu Knoten D senden. Da Knoten B auch die Daten von Knoten C empfängt, gilt der dort verwendete Kanal als belegt und es wird von Knoten B ein anderer „freier“ Kanal gewählt. Normalerweise hätte

dort für eine Kommunikation mit Knoten A der gleiche Kanal verwendet werden können, da dieser ungestört ist.

Um die oben beschriebenen Probleme zu vermeiden, nutzt beispielsweise WLAN andere Zugriffsverfahren als das beim Ethernet eingesetzte *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD). Dort werden *CSMA with Collision Avoidance* (CSMA/CA), *Request To Send/Clear To Send* (RTS/CTS) und die *Point Coordination Function* (PCF) eingesetzt. Auf detailliertere Informationen zu den Übertragungs- und Zugriffsverfahren wird an dieser Stelle verzichtet, da dies den Rahmen dieser Arbeit sprengen würde. Weitergehende Informationen liefert jedoch beispielsweise [SDHT07]. Die Zusammenhänge bezüglich der Zugriffsverfahren müssen gerade bei Ad-hoc-Netzen mit sehr vielen Netzknoten berücksichtigt werden. Dort haben sie einen wesentlichen Einfluss auf Aussagen zur Performance und zum Verkehrsverhalten, da die Anzahl der zur Verfügung stehenden Kanäle begrenzt ist.

Neben den in diesem Abschnitt erwähnten Übertragungsverfahren und Netzwerktechniken unterstützt nach [CoGi07] auch der im Standard IEEE 802.16 spezifizierte *Worldwide Interoperability for Microwave Access* (WiMAX) Ad-hoc-Netze. Es werden Reichweiten von ca. 50 km und Bitraten von bis zu 70 MBit/s unterstützt. Allerdings wurde diese Technik entwickelt, um den Nutzern einen alternativen Breitbandzugang zum Internet anbieten zu können. Dem ist auch der physische Aufbau geschuldet. In [Renh05] wird beschrieben, dass in der Referenzkonfiguration Basisstationen enthalten sind. Dies widerspricht jedoch den in Abschnitt 2.1 beschriebenen Anforderungen an ein Ad-hoc-Netz.

Zukünftig werden auch so genannte Sensornetzwerke, die ebenfalls als Ad-hoc-Netz realisiert sein können, eine große Rolle spielen. Da der Fokus bei diesen Netzen jedoch nur auf der Erfassung und Weiterleitung von Sensordaten liegt, sind sie für die Übertragung größerer Datenmengen wenig geeignet. Anwendungen, wie sie in Abschnitt 2.1 beschrieben wurden, sind damit nicht realisierbar. ZigBee (IEEE 802.15.4) ist der bekannteste Vertreter für Sensornetze. Es unterstützt Bitraten von bis zu 250 KBit/s und Reichweiten von 75 m.

Im Gegensatz dazu erscheint *Ultra Wide Band* (UWB) als Übertragungstechnik im Bereich Ad-hoc-Netze als vielversprechend. Nach [Wenz07b] soll zukünftig mit dieser Technik eine Bruttobitrate von bis zu 2 GBit/s erreicht werden, wobei die dabei mögliche Funkreichweite aufgrund der geringen zulässigen Sendeleistung auf 10 Meter begrenzt sein wird. Bedingung für einen kommerziellen Erfolg von UWB ist jedoch die Schaffung von gesetzlichen Grundlagen für einen breiten Einsatz. Gegenwärtig besteht diesbezüglich aber noch Nachholbedarf.

In Tabelle 2.3 sind die in diesem Abschnitt beschriebenen Übertragungstechniken noch einmal gegenübergestellt und deren Eigenschaften bewertet worden. Die Bewertungen (+) für positiv, (o) neutral und (–) für negativ wurden im direkten Vergleich zwischen den einzelnen Verfahren vergeben. Die Spalte „Verfügbarkeit“ bezieht sich auf die Unterstützung aktuell vorhandener Geräte. Im Gegensatz zu HIPERLAN 1/2 wird sich dies in den nächsten Monaten zwar auch für ZigBee, UWB und WiMAX verbessern, trotzdem ist auch dann die von WLAN- und Bluetooth-Produkten vorhandene Durchdringung noch lange nicht erreicht. Die Angaben unter „Reichweite“ beziehen sich auf die Werte für den Freiraum. Dort wurden die Techniken, deren Standards Reichweiten größer als 100 m spezifizieren mit (+), Reichweiten zwischen

50 m und 100 m mit (o) und Reichweiten mit weniger als 50 m mit (–) bewertet. Die Reichweiten stellen dabei gleichzeitig ein Maß für die Flexibilität eines Netzes dar. In der Spalte „Datenrate“ wurden die Bruttowerte größer als 10MBit/s mit (+), zwischen 1 und 10 MBit/s mit (o) und bei weniger als 1 MBit/s mit (–) bewertet. Schließlich beschreibt das „Kommunikationsschema“ mit (+), ob eine direkte Knoten-zu-Knoten-Übertragung möglich ist. Eine Master-Slave-Kommunikation wird mit (o) bewertet, weil dies zu einem Flaschenhals führt. Dagegen erfüllt die Kommunikation über feste Basisstationen überhaupt nicht die Anforderungen an die Mobilität und erhält deswegen ein (–). Die Auswertung der Tabelle 2.3 führt zu dem Ergebnis, dass WLAN den gestellten Anforderungen derzeit am besten entspricht und in Bezug auf verschiedene Einsatzfälle am flexibelsten ist.

Verfahren	Verfügbarkeit	Reichweite	Datenrate	Kommunikations-schema
HIPERLAN 1	–	+	+	+
HIPERLAN 2	–	+	+	o
WLAN	+	+	+	+
Bluetooth	+	o	o	o
ZigBee	o	o	–	+
UWB	–	–	+	+
WiMAX	o	+	+	–

Tabelle 2.3: Übertragungstechniken für Ad-hoc-Netze im Vergleich

2.4 Kapitelzusammenfassung

Bei Ad-hoc-Netzen handelt es sich um drahtlose mobile Netzwerke mit einer dynamischen Topologie. Durch ihre Flexibilität können sie an Orten eingesetzt werden, wo herkömmliche Infrastrukturnetze aus technischen oder ökonomischen Gründen versagen. Das eröffnet in vielen Bereichen des gesellschaftlichen Lebens neue Kommunikationsmöglichkeiten. Momentan sind die vielfältigen Einsatzmöglichkeiten von Ad-hoc-Netzen nur schwer abzuschätzen. Als sicher erscheint jedoch, dass diese Netze die Kommunikationslandschaft in den nächsten Jahren entscheidend mitprägen werden. Neben den Vorzügen gibt es auch eine Reihe von offenen Fragen sowie einige Nachteile, die die Aussage zulassen, dass sich herkömmliche drahtlose Infrastrukturnetze und Ad-hoc-Netze nicht ausschließen, sondern ergänzen werden.

Im Rahmen der vorliegenden Arbeit ist der Aufbau eines Demonstrators vorgesehen. Werden dazu die bisherigen Aussagen berücksichtigt, lässt sich zu diesem Zeitpunkt bereits ableiten, welches Übertragungsverfahren dort eingesetzt werden muss. Aktuell können Ad-hoc-Netze durch bekannte Funkverfahren wie WLAN, Bluetooth oder HIPERLAN 1/2 realisiert werden. Darüber hinaus gibt es Entwicklungen wie ZigBee, welches jedoch vorerst als Technologie für Sensornetze dient und den Anforderungen eines Ad-hoc-Netzes im Sinne dieser Arbeit nicht genügt. Ebenso verhält es sich mit WiMAX, das als Zugangstechnologie für Breitbandnetze entwickelt wurde. Auch auf UWB als Technik kann momentan nicht zurückgegriffen werden. Es handelt sich hierbei zwar um eine vielversprechende Entwicklung, aber deren lizenzrechtlichen Rahmenbedingungen sind noch nicht vollständig geklärt. Darüber hinaus

sind nicht ausreichend Hardwarekomponenten am Markt verfügbar, um ein Netzwerk überhaupt realisieren zu können. Des Weiteren entfallen HIPERLAN 1/2 als mögliche Techniken für einen Demonstrator, da sich diese Standards nicht durchsetzen konnten und auch hierfür nur vereinzelt Hardware existiert.

Letztlich verbleiben für einen Einsatz noch Bluetooth und WLAN. Hierbei ist zu berücksichtigen, dass Bluetooth eine geringere Reichweite und niedrigere Bitraten als WLAN unterstützt. Hinzu kommt, dass bei Bluetooth ein Flaschenhals durch die zwingende Kommunikation über den Master entsteht, was sich nachteilig auf den Verkehr auswirkt. Anhand der in diesem Kapitel festgestellten Eigenschaften wird deutlich, dass WLAN durch die Möglichkeit der direkten Kommunikation zweier Knoten, der problemlosen Netzerweiterung und der Unterstützung hoher Bitraten besonders für die Realisierung von Ad-hoc-Netzen geeignet ist. Dem Aufbau eines Demonstrators kommt weiterhin entgegen, dass sich WLAN in seinen verschiedenen Standardisierungen etabliert hat und vielfältige Hard- und Software unterstützend zur Verfügung stehen.

Dieses Kapitel verdeutlicht, dass die Kommunikationsabläufe in Ad-hoc-Netzen sehr komplex sind, wobei hier lediglich auf die Eigenschaften der Übertragungstechniken Bezug genommen wurde. Diese Techniken umfassen zwar keine Funktionen der Vermittlungsschicht und damit keine Routingfunktionen, schaffen aber die dafür notwendigen Voraussetzungen. Ebenso wie auf der Sicherungsschicht die Zugriffsverfahren an die speziellen Eigenschaften der Ad-hoc-Netze angepasst sind, müssen auf der Vermittlungsschicht entsprechend angepasste Verfahren zur Wegefindung bzw. zum Routing eingesetzt werden. Die Wegewahl ist primär für die Funktion eines jeden Netzes verantwortlich, wobei gerade die speziellen Eigenschaften eines Ad-hoc-Netzes besonderer Mechanismen bedürfen. Im folgenden Kapitel soll deshalb speziell auf das Problem des Routings eingegangen werden.

3. Routing in Ad-hoc-Netzen

In Kapitel 2 wurden, basierend auf der dort eingeführten Definition 2.1, bereits Ad-hoc-Netze spezifiziert und deren Funktionsweise erläutert. Dabei sind auch die verschiedenen Übertragungsverfahren beleuchtet worden. Diese Verfahren beschreiben den Datenaustausch zwischen benachbarten Netzknoten. Um aber Daten zwischen zwei Knoten austauschen zu können, die nur über weitere Zwischenknoten erreichbar sind, muss zunächst ein verfügbarer Übertragungsweg gefunden werden. Diese Funktion wird durch die Übertragungsverfahren jedoch nicht erbracht. Dafür sind Routingverfahren notwendig, die den hohen Ansprüchen der Ad-hoc-Netze genügen. Diese sollen im Folgenden untersucht werden. Der Abschnitt 3.1 führt dazu in die Thematik des Routings für Ad-hoc-Netze ein. In Abschnitt 3.2 wird auf allgemeine Probleme eingegangen, die sich aus der Komplexität der Ad-hoc-Netze ergeben. Abschnitt 3.3 beschreibt dann die Anforderungen, die die Routingverfahren erfüllen müssen. Schließlich wird in Abschnitt 3.4 ein Überblick über die wichtigsten aktuellen Routingprotokolle gegeben, während Abschnitt 3.5 auf einige spezielle Protokolle eingeht und somit die Vielfalt der Realisierungmöglichkeiten widerspiegelt. Das vorliegende Kapitel stellt insgesamt die Voraussetzung dafür dar, um ein geeignetes Routingverfahren für das kontextsensitive Routing modellieren zu können.

3.1 Einführung

Das Ziel des Routings besteht darin, einen geeigneten Weg für die zu einer Kommunikation zwischen Sender und Empfänger gehörenden Datenübertragung zu finden. Mit Hilfe entsprechender Routingprotokolle werden die Wege zu den interessierenden Zielknoten ermittelt und diese Informationen ggf. im Netz verbreitet. Die Routingprotokolle können verschiedene Kriterien verwenden, um bei einer Wahl zwischen mehreren Möglichkeiten die beste Route zu finden. Neben diesen Protokollen gibt es auch solche, die die eigentlichen Nutzdaten der Teilnehmer durch das Netz weiterleiten bzw. vermitteln. Fälschlicherweise werden solche Protokolle häufig auch als Routingprotokolle bezeichnet. Bekanntester Vertreter ist hier das IP. Dieses Protokoll der Vermittlungsschicht nutzt die durch das Routingprotokoll ermittelten Informationen, um eine Wegewahl in den Netzknoten bzw. Routern treffen zu können.

IP funktioniert diesbezüglich in Ad-hoc-Netzen genauso wie in paketvermittelten Infrastrukturnetzen. Wie bereits erwähnt, ist dies für die Routingprotokolle nicht der Fall. Eine weitere Besonderheit besteht darin, dass Routingprotokolle üblicherweise nicht in der Vermittlungs- bzw. Internetschicht angesiedelt sind, sondern Transportprotokolle nutzen und deshalb auf diese aufsetzen. In Abbildung 3.1 ist dies beispielhaft für das *Border Gateway Protocol* (BGP) dargestellt. Bezogen auf das TCP/IP-Referenzmodell ist BGP in der Anwendungsschicht angeordnet und setzt auf TCP auf.

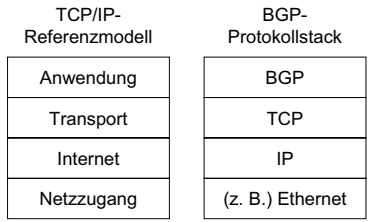


Abbildung 3.1: Einordnung des BGP in das TCP/IP-Referenzmodell

Mit den ersten Ansätzen für Ad-hoc-Netze trat sofort das Problem der Routensuche in den Vordergrund der Forschung. Dieses Thema ist auch bis heute ein Schwerpunkt vielfältiger Untersuchungen geblieben. Mittlerweile gibt es eine ganze Reihe verschiedener Lösungen dafür, wie eine passende Route für die Kommunikation zwischen zwei Netzknoten gefunden werden kann. Die Wichtigsten sollen im Folgenden vorgestellt werden. Daneben wird gezeigt, wo die Schwierigkeiten einer erfolgreichen Vermittlung liegen.

Eine Analyse der IP- oder Vermittlungsschicht in einem Ad-hoc-Netz führt zu dem Ergebnis, dass dort die gleichen Adressen genutzt werden, wie sie aus dem paketvermittelten Infrastrukturnetz bekannt sind. Da die eingesetzten Netztechniken aus Kapitel 2 den Netzzugang darstellen und Funktionalitäten der Schichten 1 und 2 des ISO/OSI-Referenzmodells realisieren, können dort die herkömmlichen Protokolle IPv4 und IPv6 aufgesetzt werden. Das kommt auch dem Wunsch der Entwickler entgegen, Ad-hoc-Netze in herkömmliche Netze einbinden zu können bzw. diese zu ergänzen. Die Datenübertragung mit diesen Protokollen funktioniert aber nur, wenn alle Knoten die Topologie des Netzes kennen und diese auf ihre Routingtabelle abbilden können. Bei Ad-hoc-Netzen mit ihrer dynamischen Topologie stellt das jedoch ein komplexes Problem dar. Deshalb können für das Erstellen der Routingtabellen nicht die bis dato aus den Infrastrukturnetzen bekannten Protokolle verwendet werden.

Bei Ad-hoc-Netzen handelt es sich um paketvermittelte Netze. Ihre Topologie ermöglicht im einfachsten Fall eine direkte Kommunikation zwischen zwei Endgeräten. Aufgrund der verwendeten Funkstandards (z. B. WLAN) besitzen die an einem Ad-hoc-Netz beteiligten Knoten jedoch eine begrenzte Reichweite, sodass nicht in jedem Fall eine direkte Kommunikation unterstützt werden kann. Deshalb sollte jedes am Netz beteiligte Gerät in der Lage sein, Anfragen und Daten anderer Teilnehmer weiterzuleiten. Jeder Knoten muss also in der Lage sein, nicht nur als Endgerät sondern auch als Relay-Station bzw. Router arbeiten zu können. Damit sind dann

auch Multihop-Übertragungen möglich, die eine paketvermittelte Kommunikation über größere Entfernungen unterstützen. Die dazu in der Vermittlungsschicht angebotenen Funktionen zur Wegewahl sind aus Sicht des TCP/IP-Referenzmodells unabhängig vom Netzzugang bzw. vom Übertragungsverfahren. Ist das Zielsystem innerhalb eines Ad-hoc-Netzes gefunden und eine Datenübertragung möglich, verhalten sich die miteinander kommunizierenden Endgeräte wie bei herkömmlichen Netzwerken.

3.2 Allgemeine Probleme

Wie in Abschnitt 2.2 bereits beschrieben, ist die Topologie von Ad-hoc-Netzen dynamisch. Das hat direkte Auswirkungen auf die Wegewahl, weil damit auch die möglichen Routen dynamisch sind. Die Routeninformationen werden innerhalb der einzelnen Knoten in den Routingtabellen abgelegt. Das heißt, dass diese Informationen je nach Mobilität der Teilnehmer veralten und dann nicht mehr die tatsächliche Struktur des Netzes repräsentieren. Deshalb muss die Wegewahl in einem solchen Netz mit Hilfe spezieller an Ad-hoc-Netze angepasster Routingverfahren erfolgen. Trotzdem arbeiten erste für solche Netze entwickelte Protokolle mit konventionellen jedoch leicht modifizierten Algorithmen. Abschnitt 3.5 wird ergänzend dazu auch einige neue Ansätze vorstellen.

Ein Aspekt, der für das Routing eine wesentliche Herausforderung darstellt, ist der Handover. Als Handover wird die Prozedur bezeichnet, bei der während einer Kommunikation zwischen zwei Endgeräten ein Umschalten von einer aktuell genutzten Route zu einer alternativen Route stattfindet. Dies wird beispielsweise dann notwendig, wenn die aktuelle Route nicht mehr aufrechterhalten werden kann, weil sich zwei an der Kommunikation beteiligte Netzknoten außer Reichweite voneinander bewegt haben. Im Idealfall merkt der Teilnehmer während seiner Kommunikation nichts davon. Da das Netz dynamisch ist und im schlimmsten Fall alle beteiligten Knoten mobil sein können, kommt es praktisch ständig zu solchen „Verbindungsübergängen“, was sehr hohe Ansprüche an das jeweilige Routingverfahren stellt, um einen nahtlosen Handover zu erreichen. Praktisch kann das gegenwärtig nur durch redundante Routen gewährleistet werden. Es sei an dieser Stelle noch erwähnt, dass neben dem Begriff Handover häufig auch der Begriff Handoff in der gleichen Bedeutung verwendet wird.

Allgemein wird zwischen vertikalem und horizontalem Handover unterschieden. Ein horizontaler Handover bezeichnet alle Handovervorgänge, die innerhalb eines Netzes erfolgen, das durch eine einzige Netzwerktechnik realisiert ist. Zum einen können hier die kommunizierenden Endgeräte durch ihre Mobilität ständig Handover initiieren. Zum anderen kommt erschwerend hinzu, dass die bei dieser Kommunikation als Router dienenden Knoten ebenfalls mobil sind. Dementsprechend treten auch Handover zwischen den Routern auf. Einen durch einen routenden Knoten verursachten Handover zeigt Abbildung 3.2, wobei dort die Route zwischen Quelle und Ziel zum Zeitpunkt t_0 und zu einem späteren Zeitpunkt t_1 dargestellt ist. Bei t_1 hat sich der betrachtete Netzknoten außer Funkreichweite der Quelle bewegt und kann nicht mehr als weiterleitender Knoten verwendet werden. Deshalb wurde ein Handover auf die untere Route notwendig. Bei den drahtlosen Infrastrukturnetzen war ein Handover lediglich auf die Beziehung Endgerät–Basistation begrenzt. Somit spielten

dort Verzögerungszeiten bzw. Umschaltzeiten nur beim Wechsel der Basisstation eine Rolle. In Ad-hoc-Netzen können zu jedem Zeitpunkt und zwischen allen Geräten Handover auftreten. Die Komplexität dieser Vorgänge steigt dementsprechend mit der Anzahl der zu einem Netzwerk gehörenden Geräte.

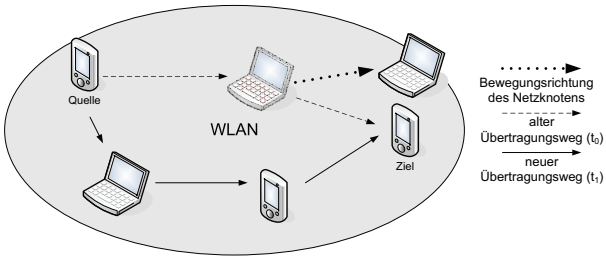


Abbildung 3.2: Beispiel eines horizontalen Handovers [SDHT07]

Ein vertikaler Handover bezieht sich auf den Wechsel zwischen verschiedenen Netztechniken (z. B. Bluetooth, WLAN, GPRS, UMTS). Das ist beispielsweise der Fall, wenn sich ein Teilnehmer aus dem Bereich eines Bluetooth-Netzes herausbewegt, aber auf eine WLAN-Umgebung zurückgreifen kann und automatisch dort eingebucht wird. Bei einem solchen Handover ist dann nicht ausgeschlossen, dass selbst ein Wechsel des Backbone-Netzes nötig wird, da sich die Kommunikation, wie in Abbildung 3.3 dargestellt, zwischen zwei Teilnehmern über mehrere Netztechniken erstrecken kann.

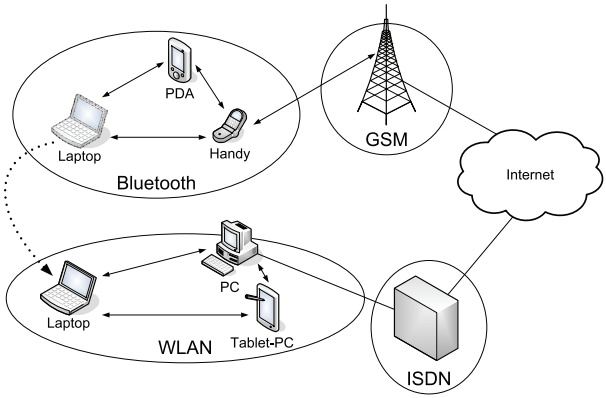


Abbildung 3.3: Vertikaler Handover mit Wechsel des Backbone-Netzes [SDHT07]

Wie in [Nowa06] nachgewiesen wurde, haben Handover einen starken Einfluss auf Multimedia- und Echtzeitanwendungen. Natürlich muss hierbei berücksichtigt werden, dass die Messungen nur mit einem Routingprotokoll – dem *Optimized Link State Routing* (OLSR) – durchgeführt wurden. Es können damit aber auch durchaus allgemeine Aussagen bezüglich des Einflusses von Handovern mit Hilfe tabellenbasierter Routingverfahren auf die Echtzeitfähigkeit eines Ad-hoc-Netzes getroffen

werden. Für die Messungen diente als Netzwerkzugang WLAN nach IEEE 802.11b. Ein Ergebnis der Arbeit war, dass mit einem solchen Routingprotokoll derzeit keine oder nur in geringem Maße Multimedia- und Echtzeitanwendungen unterstützt werden. Das liegt vor allem am Alterungsprozess der Routingtabellen und der deshalb notwendigen Aktualisierungen. Die Echtzeitfähigkeit kann durch Netzwerktechniken mit höherer Bitrate zwar verbessert werden, anderseits würde sich das Verhalten beispielsweise durch eine zunehmende Mobilität der Knoten verschlechtern. Die Messungen wurden nur für den horizontalen Handover durchgeführt. Jedoch lassen sich damit auch Schlussfolgerungen für einen vertikalen Handover ziehen. Da dort zusätzlich auch Verzögerungseffekte durch den Wechsel der Netzwerktechnik und den Zugang zum alternativen Netz hinzukommen, kann das Verhalten bezüglich Multimedia- und Echtzeitanwendungen nur noch schlechter werden. Deshalb sollten gerade beim vertikalen Handover Techniken verwendet werden, die die Verzögerungszeit im Bereich des Netzzuganges minimieren. Erreicht werden kann das beispielsweise, indem mehrere alternative Netzzugänge parallel offen gehalten werden oder der Zeitpunkt zu einem alternativen Netz so frühzeitig gewählt wird, dass die Routensuche schon vor dem anstehenden Handover beendet werden kann. Die hier angedeutete Komplexität der Handovervorgänge zeigt deutlich den großen Forschungsbedarf im Bereich des Handovers bei Ad-hoc-Netzen.

Problematisch ist neben dem eigentlichen Handover die Gewährleistung der Interoperabilität zwischen den verschiedenen Netzen und (Routing-)Protokollen. Zu berücksichtigen sind unter anderem unterschiedliche Adressierungsverfahren und Bitraten sowie die mehr oder weniger gute Unterstützung von QoS-Parametern. Während bei Ad-hoc-Netzen paketorientierte Vermittlungsverfahren verwendet werden, sind es beispielsweise bei den klassischen Telefonnetzen leitungsorientierte Vermittlungsverfahren. Selbst die Zusammenarbeit mit den aus dem Internet bekannten Protokollen ist auf Grund der dynamischen Netzstruktur nicht unproblematisch. Auch diese Themen werden gegenwärtig noch untersucht.

Zu den in diesem Abschnitt beschriebenen Herausforderungen kommen auch noch die in Kapitel 2 dargestellten Eigenschaften der beiden unteren Protokollschichten hinzu. Dort auftretende Übertragungsstörungen wirken sich direkt auf die Vermittlungsschicht aus. Fällt z. B. ein routender Knoten wegen eines leeren Akkus aus, muss für alle darüber ausgeführten Datenübertragungen eine neue Route zwischen Quelle und Ziel gefunden werden. Welche Anforderungen sich aus den dargestellten Problemen ergeben, zeigt der folgende Abschnitt.

3.3 Anforderungen an Routingverfahren

Werden die in den Abschnitten 3.1 und 3.2 diskutierten Eigenschaften von Ad-hoc-Netzen berücksichtigt, muss für eine korrekte Funktion der dort eingesetzten Routingprotokolle und -algorithmen eine Vielzahl von Anforderungen erfüllt sein. Hierbei kann zwischen unbedingten Anforderungen, die der Funktionsfähigkeit dienen, und optionalen Anforderungen, die die Routingfunktionalität optimieren, unterschieden werden. Diese Differenzierung spiegelt auch Tabelle 3.1 wider.

Zur ersten Kategorie gehört beispielsweise die Schleifenfreiheit. Der Routingalgorithmus muss also so beschaffen sein, dass eine Route durch das Netz in jedem Fall endlich ist. Gleichzeitig muss eine stabile Kommunikation gewährleistet werden, d. h., dass die Verbindung für die Zeit der Kommunikation aufrecht erhalten

wird. Wichtig ist auch, dass die Teilnehmer eindeutig adressiert werden können. Und schließlich trägt auch ein Mindestmaß an Sicherheit (z. B. Schutz vor Manipulation der Routinginformationen) zu einer einwandfreien Funktionsweise bei. Ohne das Erfüllen der beschriebenen Anforderungen wäre eine Wegefindung in einem Netzwerk praktisch unmöglich.

In die Kategorie der Optimierung von Funktionalitäten ordnen sich z. B. Mechanismen zur Lastverteilung ein. Weitere Beispiele hierfür wären Routingprotokolle, die auch erweiterte Sicherheitsaspekte wie Vertraulichkeit o. ä. berücksichtigen, oder Verfahren, die Rücksicht auf den Energieverbrauch nehmen, da in der Regel gerade bei Ad-hoc-Netzen jeder Knoten seine eigene unabhängige Energiequelle mitbringt. Auch die Unterstützung von QoS-Parametern[1] stellt eine Optimierung der Netzfunktionen dar. Wie bereits diskutiert, ist für Ad-hoc-Netze die Fähigkeit Handover zu unterstützen ebenfalls von großer Bedeutung. Des Weiteren muss in vielen Fällen eine Kommunikation zwischen heterogenen Netzen unterstützt werden, d. h. die Interoperabilität zwischen den Netzen ist zu optimieren. In die hier diskutierte Kategorie passen auch jegliche Mechanismen, die allgemein eine höhere Stabilität der Kommunikation gewährleisten. Letztlich sind ggf. auch Eigenschaften wie Multicastfähigkeit, geringer Overhead o. ä. zu berücksichtigen. Weitere Beispiele sind Tabelle 3.1 zu entnehmen, wobei die dortige Einteilung keine Auskunft darüber gibt, mit welcher Qualität die einzelnen Anforderungen tatsächlich umgesetzt werden können.

Unbedingte Anforderungen	Optionale Anforderungen
• Schleifenfreiheit	• Lastverteilung
• hohe Stabilität der Kommunikation	• Vertraulichkeit
• eindeutige Adressierung	• niedriger Energieverbrauch
• Sicherheit	• geringer Overhead
	• dienstgütebehafteter Handover
	• Dienstlokalisierung
	• Unterstützung mobiler Dienste
	• Unterstützung von Multicast
	• Kompatibilität zu anderen Netzen
	• Minimierung der Hopanzahl
	• QoS-Parameter

Tabelle 3.1: Anforderungen an Routingprotokolle [SDHT07]

Wird die Liste der Anforderungen analysiert, bildet sich möglicherweise der Wunsch heraus, ein Routingprotokoll zu entwickeln, das die meisten oder sogar alle Anforderungen erfüllt. Dieser Wunsch scheitert jedoch an einem Widerspruch. Je mehr Funktionen ein Protokoll unterstützt bzw. je mehr Anforderungen erfüllt werden, desto mehr Informationen müssen durch das Netzwerk transportiert werden. Der Overhead steigt also mit der Menge der Funktionen an. Das wiederum schränkt die eigentlich für Daten zur Verfügung stehende Bitrate ein. Des Weiteren können sich verschiedene Anforderungen auch gegenseitig ausschließen. So kann beispielsweise die Wahl einer Route mit möglichst wenigen Hops zu störanfälligen und damit

[1] Als QoS ist an dieser Stelle nicht das grundsätzliche Aufrechterhalten einer Kommunikation beispielsweise durch alternative Routen gemeint, sondern die Garantie von Bitrate, Laufzeit, minimaler Bitfehlerrate usw.

zu unzuverlässigen Übertragungen führen, da sich durch die größeren Funkstrecken das Signal-Rausch-Verhältnis verschlechtert. Als Erkenntnis aus diesen Aussagen lässt sich schlussfolgern, dass bei der Entwicklung von Routingprotokollen immer ein Kompromiss zwischen Protokollfunktionalität, Anforderungen und Größe der Overheaddaten gefunden werden muss. Es ist somit äußerst unwahrscheinlich, dass es zukünftig ein Protokoll oder ein allgemeingültiges Routingverfahren geben wird, das alle Anforderungen erfüllen kann. Es wird auf der einen Seite immer Routingprotokolle geben, die nur wenige Anforderungen erfüllen, dafür aber ein breites Einsatzgebiet unterstützen. Auf der anderen Seite werden parallel dazu auf verschiedene Netze und Anwendungsfälle hin optimierte Routingprotokolle existieren, was deren gegenwärtige Anzahl auch beweist.

Die in diesem Abschnitt aufgezählten Anforderungen stellen eine repräsentative Übersicht dar, die die wichtigsten technischen Fähigkeiten von Routingverfahren betreffen. Darüber hinaus ist die Spezifikation zusätzlicher Anforderungen möglich. So können beispielsweise auch wirtschaftliche Aspekte (z. B. Kosten einer Route) eine Rolle spielen. Auf eine vollständige Aufzählung muss jedoch verzichtet werden, weil sich durch die stetige Entwicklung neuer Einsatzzwecke im Bereich der Routingprotokolle auch neue Anforderungen ergeben. Beispielsweise können Fähigkeiten, wie das Lokalisieren von Diensten bzw. die Unterstützung mobiler Dienste, die bisher kaum eine Rolle gespielt haben, an Bedeutung gewinnen und für bestimmte Anwendungsfälle unerlässlich sein.

3.4 Übersicht über aktuelle Routingverfahren

Um ein Paket in einem Netz weiterleiten zu können, muss der betreffende Knoten Informationen darüber besitzen, wie das gewünschte Ziel erreicht werden kann. Die Router nutzen hierfür drei Arten von Informationen. Es werden nach [ChMi01] Informationen unterschieden, die den lokalen Status (*local state*), den globalen Status (*global state*) oder den aggregierten (zusammengefassten) globalen Status (*aggregated global state*) beschreiben. Der *local state* umfasst Informationen über das eigene Gerät und seinen unmittelbaren Nachbarn. Dazu gehören beispielsweise die verfügbare CPU-Kapazität, Verzögerungen durch Warteschlangen, Aussagen über die Bandbreite sowie über die Metrik abgehender Links. Diese Informationen sind immer verfügbar. Der *global state* dagegen umfasst Informationen über das gesamte Netz. Hierzu gehören z. B. der Verbindungsstatus und verschiedene Informationen aus dem *local state* der im Netz beheimateten Geräte. Um diese Informationen auf dem aktuellen Stand zu halten, sind in bestimmten Abständen Updates nötig. Der Abstand zwischen zwei Updates muss an die Dynamik des Netzes angepasst sein, um den Fehler zwischen gespeicherter Information über den Status des Netzes und dem tatsächlichem Zustand möglichst gering zu halten. Jedes Update wirkt sich jedoch negativ auf die Verkehrslast aus. Hier ist ein Kompromiss zwischen möglichst hoher Genauigkeit der Informationen und geringer Erhöhung der Verkehrslast zu suchen. Der aggregierte globale Status spielt bei hierarchischen Netzen eine Rolle. Ist ein Netz in so genannte Cluster aufgeteilt, werden hier die globalen Informationen über den Cluster gesammelt. Welche Informationsarten und welche Informationen benötigt werden, hängt vom verwendeten Routingprotokoll ab.

Erste Routingverfahren und -protokolle, die für Ad-hoc-Netze entwickelt wurden, sind stark an die Protokolle der paketvermittelten Infrastrukturnetze angelehnt. Das zeigt auch die Gegenüberstellung in Abbildung 3.4. Hier werden die Routingprotokolle in zwei Gruppen basierend auf der Methode des Managements ihrer Routinginformationen gegliedert. [Toh02] teilt die Protokolle dazu in *Table Driven*, *Source-initiated On-Demand Driven* und *Hybrid* ein.

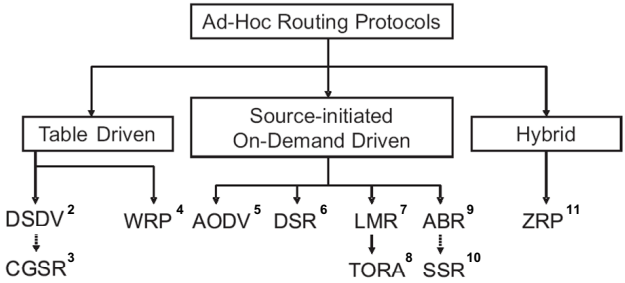


Abbildung 3.4: Klassische Ad-hoc-Netz-Routingprotokolle [Toh02]

Tabellengesteuerte (*Table Driven*) Routingverfahren basieren auf dem Distance Vector- bzw. Link State-Routingalgorithmus. Es handelt sich hierbei um proaktive Verfahren, bei denen die Routinginformationen vor der eigentlichen Benutzung einer Route gesammelt und bei Bedarf sofort darauf zugegriffen werden kann. Abgelegt werden diese Informationen in den Routingtabellen der jeweiligen Knoten. Hierbei muss jedoch das so genannte *Table Aging* berücksichtigt werden. Die Routinginformationen werden aufgrund des dynamischen Charakters des Ad-hoc-Netzes mit zunehmendem Alter ungenauer. Deshalb müssen, wie bereits bei den Statusinformationen beschrieben, die Routingtabellen in bestimmten Zeitabständen aktualisiert werden. Der Zusammenhang zwischen Genauigkeit der Routinginformationen und Höhe der Verkehrslast führt zu zwei Erkenntnissen. Erstens muss bei der Verwendung von *Table Driven*-Verfahren ein geeignetes Optimum zwischen dem zeitlichen Abstand zweier Updates und der durch die Updates verursachten aber noch zu akzeptierenden Verkehrslast gefunden werden. Und zweitens sind solche Algorithmen für hochgradig mobile Teilnehmer überhaupt nicht geeignet. Hier muss nach anderen Lösungen gesucht werden.

Bei den bedarfsgesteuerten (*Source-initiated On-Demand Driven*) Routingverfahren handelt es sich um reaktive Mechanismen, bei denen die Route zum Ziel erst dann ermittelt wird, wenn sie der Sender wirklich benötigt. Diese Informationen werden

[2]DSDV *Destination-Sequenced Distance-Vector*
[3]CGSR *Cluster-Head Gateway Switch Routing*
[4]WRP *Wireless Routing Protocol*
[5]AODV *Ad hoc On-Demand Distance Vector*
[6]DSR *Dynamic Source Routing*
[7]LMR *Lightweight Mobile Routing*
[8]TORA *Temporally-Ordered Routing Algorithm*
[9]ABR *Associativity-Based Routing*
[10]SSR *Signal Stability Routing*
[11]ZRP *Zone Routing Protocol*

ebenfalls in entsprechenden Routingtabellen abgelegt und müssen, solange sie gültig sind, in bestimmten Zeitabständen aktualisiert werden. Ein wesentlicher Nachteil dieser Verfahren besteht in der „verlorenen“ Zeit, die von der Suche einer geeigneten Route bis zum ersten Datenaustausch vergeht. Der Vorteil besteht in der Aktualität der nutzbaren Routinginformationen. Im schlechtesten Fall wird das Netz bei diesem Verfahren mit einer Wegesuchanfrage geflutet, wodurch alle Geräte im Netz ein *Request* erhalten und ein großer Anstieg der Verkehrslast zu verzeichnen ist. Danach gelten für die Aktualisierung der Routeninformationen im Wesentlichen die gleichen Bemerkungen wie oben.

Es gibt auch Ansätze, die die beiden eben beschriebenen Verfahren kombinieren. Ein Beispiel hierfür wäre das *Zone Routing Protocol* (ZRP [HaPS02]). Bei diesem Protokoll hat jeder Knoten Kenntnisse über die Netzwerktopologie in seiner unmittelbaren lokalen Region – seiner spezifischen *Routing Zone*. Die dazu notwendigen Routinginformationen werden proaktiv ermittelt. Eine Routensuche zu einem Knoten außerhalb der Zone basiert dagegen auf dem reaktiven Ansatz und erfolgt somit erst bei Bedarf.

Ein Problem für sämtliche Verfahren besteht darin, ihre zum Routing notwendigen Informationen ständig zu aktualisieren. Bei Ad-hoc-Netzen nimmt die Komplexität mit größer werdenden Netzwerken stark zu. Das bedeutet auch, dass eine mit der Anzahl der Netzknoten wachsende Bitrate benötigt wird, um Update-Informationen auszutauschen. Das ZRP begrenzt z. B. den Aufwand für das Management von Routinginformationen, indem Informationen nur über solche Knoten gesammelt werden, die in der eigenen Zone liegen. Damit müssen nur die Routen innerhalb dieser Zone aktualisiert werden. Informationen zu außerhalb liegende Knoten werden erst bei einem konkreten Kommunikationswunsch ermittelt und sind nur für die Dauer der Kommunikation valid.

Ein weiterer bekannter Ansatz ist, das Netz wie beim *Cluster Based Routing Protocol* (CBRP [JiLi99]) in Cluster einzuteilen. Hier gehört jeder Knoten zu einer bestimmten Gruppe (dem Cluster). Die Knoten innerhalb eines Clusters können sich alle untereinander erreichen. Einer der Knoten (der *Cluster Head*) kennt alle Mitglieder des Clusters und besitzt auch Informationen über die Routen zu anderen Clustern. Jeder Knoten besitzt Informationen darüber, welche Nachbarn sich im Umkreis von zwei Hops befinden, unabhängig davon, zu welchem Cluster sie gehören. Damit ist zwar das Wissen der einzelnen Knoten aber auch der nötige Informationsaustausch für Routing-Updates begrenzt. Das spart Bitrate und der Verwaltungsaufwand verringert sich.

Die Zahl der Routingprotokolle wächst ständig weiter. Eine umfangreiche Liste liefert hier die Enzyklopädie Wikipedia [AdhR07]. Tabelle 3.2 spiegelt dagegen einen repräsentativen Überblick über die aktuell wichtigsten Routingprotokolle und deren Funktions- bzw. Arbeitsweise wider. Als Basis dieser Tabelle diente [Born01]. Die dort verglichenen Routingverfahren wurden von [Kram05] übernommen, aktualisiert und erweitert. Trotzdem waren die Aussagen bezüglich der angegebenen Eigenschaften noch sehr lückenhaft. Die Tabelle 3.2 wurde nun im Rahmen dieser Arbeit ergänzt und vervollständigt. Sie spiegelt den aktuellen Stand wider. Außerdem ist sie um das Spine-Routing erweitert worden. Die Bedeutung der einzelnen Abkürzungen sind aufgelistet. Die wichtigsten Quellen, die die Aussagen in der Tabelle belegen, sind ebenfalls angegeben. Die Tabelle selbst beschreibt die Art der Routenbestim-

mung und trifft Aussagen darüber, ob das jeweilige Protokoll QoS unterstützt, wie die Verwaltung der Routinginformationen erfolgt und ob Multicastkommunikation möglich ist. Da alle aufgelisteten Protokolle über längere Beobachtungszeiträume schleifenfrei sind, wurde auf eine explizite Angabe verzichtet. Einzelne in der Tabelle aufgelisteten Eigenschaften konnten nicht immer zweifelsfrei bestätigt werden, weil dazu aktuell keine aussagekräftigen Informationen existieren bzw. veröffentlicht sind. Die Abkürzung k. A. („keine Angabe“) kennzeichnet dies.

Routing-verfahren	Routen-bestimmung	QoS	flach/ hierarchisch	Multi-cast	Nachweise / Quellen
ABR	reaktiv	ja	flach	nein	[Toh99]
AODV	reaktiv	ja[20]	flach	ja[20]	[BRPD03], [PeBR01], [RoPe99]
CBRP[12]	hybrid	nein	hierarchisch	nein	[JiLi99]
CEDAR[13]	hybrid	ja	hierarchisch	ja[20]	[SiSB98], [SiSB99]
CGSR	proaktiv	nein	hierarchisch	ja[20]	[CCGe97], [ChGZ98]
DSDV	proaktiv	ja[20]	flach	nein	[PeBh94], [ChTG97], [JaWu05]
DSR	reaktiv	ja[20]	flach	ja[20]	[JoHM07], [LLPC+01], [JHMJ01]
FSR[14]	proaktiv	ja	hierarchisch	k. A.	[GePH02]
LMR	reaktiv	nein	flach	nein	[CoEp95]
OLSR[15]	proaktiv	ja[20]	flach	ja[20]	[JaCl03], [BaAg07], [LeCF06], [JMLV+01]
PARO[16]	reaktiv	ja[20]	flach	k. A.	[GCNB01], [Gome02]
RDMAR[17]	reaktiv	ja	flach	ja	[AgTa99b], [AgTa99a]
Spine	hybrid	nein	hierarchisch	ja	[SiDB98]
SSR	reaktiv	nein	flach	nein	[DRWT97]
STAR[18]	proaktiv	nein	flach	k. A.	[AcSB99]
TBRPF[19]	proaktiv	nein	flach	ja[20]	[OgTL04], [Ogie01]
TORA	reaktiv	ja[20]	flach	ja[21]	[PaCo01], [LeCa98]
WRP	proaktiv	nein	flach	ja[20]	[MuGLA96], [LSHG+00]
ZRP	hybrid	ja	hierarchisch	ja[20]	[HaPS02], [ZhJa04]

Tabelle 3.2: Vergleich verschiedener Routingprotokolle

[12] CBRP *Cluster Based Routing Protocol*
[13] CEDAR *Core Extraction Distributed Ad hoc Routing*
[14] FSR *Fisheye State Routing*
[15] OLSR *Optimized Link State Routing*
[16] PARO *Power-Aware Routing Optimization*
[17] RDMAR *Relative Distance Micro-discovery Ad hoc Routing*
[18] STAR *Source Tree Adaptive Routing*
[19] TBRPF *Topology Dissemination Based On Reverse-Path Forwarding*
[20] mit Erweiterung bzw. Aufsatz
[21] Voraussetzung IMEP (*Internet MANET Encapsulation Protocol*)

3.5 Spezielle Routingverfahren

In der Regel nutzen Routingverfahren bei der Auswahl eines möglichen Weges die Variante mit der kürzesten Strecke. Daneben gibt es noch eine Vielzahl anderer Routingverfahren, die auf spezielle Anforderungen bzw. Anwendungsfälle hin entwickelt wurden. Eine kleine Auswahl an Anforderungen zeigt die folgende Aufzählung:

- geringste Kosten – Kosten können hierbei über Last, Bitrate, tatsächliche Kosten o. ä. bewertet werden.
- robustester/widerstandsfähigster/stärkster Pfad – kann als Maß der Zuverlässigkeit benutzt werden.
- geringster Energieaufwand – es wird die Route verwendet, bei der der routende Knoten die geringste Sendeleistung benötigt.
- positionsbasiert – da die Position der einzelnen Knoten bekannt ist, kann daraus eine optimale Route ermittelt werden.
- QoS – der Weg wird entsprechend der QoS-Anforderungen gewählt.

Des Weiteren sind natürlich auch solche Routingverfahren vorstellbar, die gleichzeitig mehreren oben genannten Anforderungen gerecht werden. Bei genauerer Betrachtung dieser Routingprotokolle ist festzustellen, dass diese für die Verwendung in Ad-hoc-Netzen teilweise besser geeignet sind als die in Abschnitt 3.4 beschriebenen. So sind hier vor allem positionsbasierte Verfahren sinnvoll, da sich durch die dynamische Struktur des Netzes die Position der einzelnen Knoten ständig ändert. Optimale Routen können jedoch nur gefunden werden, wenn die Positionen der einzelnen Knoten bekannt sind. So dient z. B. beim *Position-based Routing in Ad Hoc Networks* [Stoj02] die Position der zum Netz gehörenden Knoten zur Topologiebeschreibung und Routenfindung. Diese Verfahren arbeiten wiederum nach verschiedenen Ansätzen. Beispielsweise wird mit relativen Positionsangaben oder direkt mit Daten aus dem *Global Positioning System* (GPS) gearbeitet. Für heterogene Netze sind solche Verfahren weniger geeignet, da die Knoten in Infrastrukurnetzen entweder ortsgebunden sind oder aber über eine Basisstation kommunizieren. Ein positionsbasiertes Verfahren erbringt hier keine Vorteile gegenüber herkömmlichen Routingprotokollen.

Mobile Knoten benötigen eine unabhängige Energiequelle, weswegen diese Geräte in der Regel über Akkus versorgt werden. Je mehr Rechenleistung ein Knoten benötigt, desto mehr Energie wird verbraucht und die Betriebsdauer nimmt ab. Das heißt, der Energieverbrauch nimmt mit rechenintensiven Routingverfahren, durch ständige Routinganfragen bzw. durch sogenannte „Leuchtfeuer“ (Beacons) und auch beim eigentlichen Weiterleiten von Paketen, zu. Deshalb wurden auch schon Verfahren entwickelt, die dahingehend optimiert sind, dass beim Routing möglichst wenig Energie verbraucht wird. Dies wird als *Power-Efficient Routing* oder batteriesparendes Routing bezeichnet. Hierzu gehören beispielsweise das *Conditional Max-Min Battery Capacity Routing* (CMMBCR) [Toh01] und das *Power-Aware Routing Optimization* (PARO).

Wieder andere Routingprotokolle sind auf die Zusammenarbeit mit verschiedenen Netzwerken optimiert. So wird beispielsweise versucht mit Hilfe des *Region-Based*

Routings (RBR) [Teng06] Ad-hoc-Netze mit dem Internet zu vereinen. Der Routingalgorithmus betrachtet zur Routenfindung nur kleine topologische Bereiche. Der Overhead wird damit gegenüber flooding-basierten Routingalgorithmen verringert. Jedoch wird auch hier kein einheitliches Protokoll für das hybride Netz verwendet. Im Bereich des Festnetzes kommen herkömmliche Routingprotokolle zur Anwendung.

Die Liste der Protokolle, die für spezielle Anwendungsbereiche entwickelt wurden, könnte noch erweitert werden. Die hier gezeigte Auswahl soll lediglich den aktuellen Trend darstellen und die wichtigsten Entwicklungsrichtungen aufzeigen. Dem entspricht auch die Klassifizierung aktueller Routingprotokolle in [CoGi07]. Dort werden neben proaktiven, reaktiven und hybriden Protokollen sowie hierarchischen Routingverfahren positionsbasierte und energieoptimierte Protokolle identifiziert.

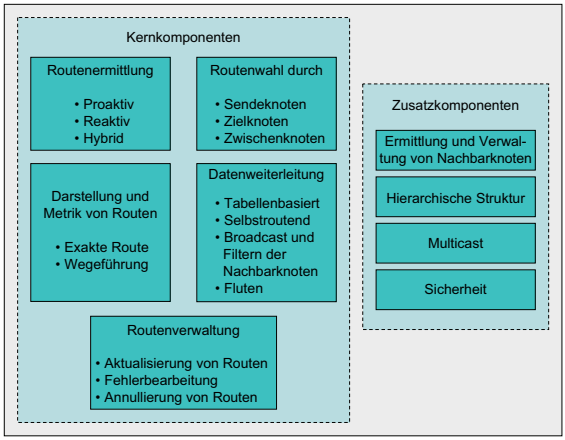


Abbildung 3.5: Routingkomponenten [LZHJ+06]

Für Ad-hoc-Netze werden ebenso wie für herkömmliche Netzwerke immer neue Dienste entwickelt. Des Weiteren werden die angebotenen Dienstfunktionen immer komplizierter. Deshalb wird aktuell versucht, verschiedene Routingansätze so zu kombinieren, dass die Anforderungen für die jeweilige Anwendung erfüllt werden. Eine solche Flexibilität soll mit dem komponentenbasierten Ansatz erreicht werden. In [LZHJ+06] wird dazu das *Component-Based Routing Protocol* (CBR) vorgeschlagen. Die zugehörige Taxonomy ist in Abbildung 3.5 dargestellt. Bei diesem Konzept werden die Funktionalitäten des Protokolls in Blöcke untergliedert und zwischen Kern- und Zusatzkomponenten unterschieden. Die Kernkomponenten enthalten die Funktionalitäten, die unbedingt von einem Routingprotokoll gefordert werden. Die Zusatzkomponenten werden zwar für eine Funktion von Routingprotokollen nicht unbedingt benötigt, verbessern aber deren Performance. Innerhalb dieser Komponenten werden die Funktionen in weiteren Blöcken untergliedert. Letztlich wird die Flexibilität dieses Ansatzes dadurch erreicht, dass alle Blöcke mit verschiedenen Funktionalitäten belegt und miteinander kombiniert werden können. Damit soll eine optimale Anpassung an gegebene Anforderungen ermöglicht werden. Inwieweit ein solcher Ansatz tatsächlich umgesetzt werden kann, bleibt abzuwarten, da sich das CBR noch in der Entwicklung befindet.

3.6 Kapitelzusammenfassung

Aufgrund der Aktualität des Themas und der daraus resultierenden vielfältigen Forschungen existiert eine große Menge an Routingprotokollen. Damit wird auch die Bedeutung der Thematik widergespiegelt. Das vorliegende Kapitel kann nur einen Überblick über die wichtigsten Routingverfahren und -protokolle für Ad-hoc-Netze geben. Diese Protokolle und aktuelle Entwicklungstrends wurden gegenübergestellt. Anforderungen, die an die Routingprotokolle gestellt werden können, wurden diskutiert und mögliche Probleme in der Funktionsweise analysiert. Neben den aufgelisteten proaktiven, reaktiven und hybriden Routingverfahren ist auch auf solche eingegangen worden, die auf spezielle Anforderungen bzw. Anwendungsfälle hin entwickelt wurden. Trotz der Protokollvielfalt bleiben aber weiterhin viele Fragen offen, die beispielsweise die Abrechnung für eine Datenübertragung oder Sicherheitsaspekte betreffen. Auch muss festgestellt werden, dass hochgradig mobile Teilnehmer mit den gegenwärtigen Routingalgorithmen nur in geringem Maß oder überhaupt nicht unterstützt werden.

Es wurde die Komplexität des Routings innerhalb eines Ad-hoc-Netzes aufgezeigt und darauf eingegangen, welche Besonderheiten zu berücksichtigen sind. Da ein solches Routingverfahren die Anforderungen eines Ad-hoc-Netzes erfüllen muss, kann es wiederum die Basis einer Kommunikation in jeglichem Netzwerk darstellen und damit die Grundlage für das kontextsensitive Routing bilden. Zwei Punkte sollten bei der Auswahl eines Routingprotokolls auf jeden Fall berücksichtigt werden:

1. Es muss ein Protokoll gewählt werden, was ein möglichst breites Anwendungsspektrum unterstützt, um die Funktion des Ad-hoc-Netzes im Sinne von Kapitel 2 zu gewährleisten.

2. Es müssen die Anforderungen, die sich aus dem kontextsensitiven Routing ergeben, erbracht werden.

Um die im zweiten Punkt genannten Anforderungen identifizieren zu können, ist zunächst zu untersuchen, was zu einer kontextsensitiven Diensterbringung notwendig ist, da diese durch das kontextsensitive Routing unterstützt werden soll. Das folgende Kapitel widmet sich dieser Aufgabe.

4. Kontextsensitive Diensterbringung

Zum besseren Verständnis der Notwendigkeit für die Entwicklung des kontextsensitiven Routings wird in diesem Kapitel zunächst der Begriff Kontext definiert und beschrieben, wie dieser mit Hilfe von Kontexttypen klassifiziert werden kann. Nach einer Einführung in die Thematik wird diskutiert, welche Informationen benötigt werden, um den Nutzerkontext beschreiben zu können. Danach wird darauf eingegangen, wie dieser Kontext erfasst und die sich daraus ergebenden Kontextinformationen für die Anwendungen aufbereitet werden können. Es folgt eine Diskussion über den in dieser Arbeit gebrauchten Dienstbegriff. Anschließend werden aktuelle Architekturen und Systeme, die kontextsensitive Dienste unterstützen, gegenübergestellt und deren Defizite aufgezeigt.

4.1 Einführung

Wie in Abschnitt 1.1 bereits festgestellt wurde, ist die Evolution der Netze unmittelbar mit dem Entstehen neuer Dienste verbunden. Gegenwärtig erfolgt eine stetige Integration aller Kommunikationsnetzwerke. Diese hat zum Ziel, einem Nutzer den Zugriff zu einer Vielzahl von Diensten unabhängig von Ort und Zeit zu ermöglichen. In diesem Zusammenhang ist häufig auch die Rede vom *pervasive and ubiquitous* (durchdringenden und allgegenwärtigen) *Computing*, dessen Entwicklung schon durch [Weis91] vorhergesagt wurde. Dem Nutzer stehen in diesem Szenario überall (z. B. integriert in Kleidungsstücken, Schmuckwaren, Kfz, Werkzeugen, ...) Prozessoren bzw. Computer jeglicher Art und Größe zur Verfügung. Diese unterstützen ihn bei der Nutzung verschiedenster Dienste. D. h., dass zukünftig Dienste netzwerkübergreifend in einem System von dezentralen Computern angeboten werden. Dem Nutzer selbst sollen diese Systeme verborgen bleiben.

Neben der Möglichkeit, Dienste überall und zu jeder Zeit nutzen zu können, werden diese auch immer individueller angeboten. Es erfolgt also eine Personalisierung des Dienstangebotes. Damit wird versucht, auf individuelle Bedürfnisse des Nutzers angepasste Dienste optimal bereitzustellen. Diese müssen dazu beispielsweise auf Informationen zugreifen können, die Auskunft darüber geben, in welchem Umfeld sich

der Nutzer befindet. Je mehr Informationen über den Nutzer bzw. dessen Kontext also bekannt sind, desto besser können bestimmte Dienste auf ihn angepasst werden. Allgemein werden solche Dienste, die abhängig von einem Kontext erbracht werden, als kontextsensitive Dienste bezeichnet. Was hierbei unter Kontext zu verstehen ist, diskutiert der folgende Abschnitt.

4.2 Kontext

Eine Beobachtung der aktuellen Entwicklung des Angebotes von Diensten lässt folgenden Trend sehr deutlich erkennen. Zukünftig werden neben den allgemeinen Diensten vermehrt auch Dienste angeboten, die speziell auf den Dienstnutzer angepasst sind. Mit Anpassung sind hierbei nicht nur die Bedürfnisse des einzelnen Individiums gemeint, vielmehr wird auch dessen aktuelle Situation bzw. Kontext berücksichtigt. Dienste, die diesen Kontext einbeziehen, sind die so genannten kontextsensitiven Dienste. Der Begriff „Kontext“ ist vielschichtig und wird sehr unterschiedlich gebraucht. Deshalb soll an dieser Stelle eine Definition für den Gebrauch in dieser Arbeit eingeführt werden. Dieses Gebiet ist aktuell Gegenstand vielfältiger Untersuchungen. So wird in [DeLS05] ausführlich darüber diskutiert, welche Definition den Begriff Kontext am besten erklärt. Ein Ergebnis ist dabei, dass die Nutzung des Begriffes sehr stark von dessen Verwendungszweck und Einsatzgebiet abhängt. So wird Kontext beispielsweise an bestimmte Eigenschaften gebunden. In [BrBC97] sind dies die Position des Nutzers, die Identifikation von Personen um den Nutzer herum, die Tageszeit, die Jahreszeit sowie die Temperatur. [RyPM97] definiert den Kontext ähnlich. Zur Beschreibung des Kontextes dienen hier aber nur Nutzerposition, Umgebung, Identität und Zeit. In beiden Fällen werden zur Erklärung des Kontextes jedoch lediglich Kontexttypen verwendet, anstatt tatsächlich eine allgemeine Definition zu bieten. Weiter gefasst ist der Begriff dagegen in [SKATL+99], weil hier neben den schon erwähnten Eigenschaften auch die Situation und der Status des Nutzers mit berücksichtigt wird. Noch allgemeiner ist die Aussage von [PaRM98], wo der Kontext als Teilmenge eines physischen oder konzeptionellen Zustandes beschrieben wird. Darauf bauen auch verschiedene Arbeiten von A. Dey und G. Abowd auf (z. B. [DeAb99, Dey00a, Dey00b]), in welchen der Begriff des Kontextes weiterentwickelt wurde. Im Ergebnis ergab sich für A. Dey schließlich in [Dey01] folgende Variante:

Definition 4.1 (Kontext) *„Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.“*

Aus dieser Definition ergeben sich folgende Schlussfolgerungen. Kontext umfasst bestimmte Informationen, die die Situation einer betrachteten Person, eines Ortes oder Objektes beschreiben. Hierbei sind jedoch nicht wirklich alle dieser Entitäten von Relevanz. In Betracht kommen nur solche, die für die aktuelle Interaktion zwischen Nutzer und Anwendung von Bedeutung sind. Der Kontext kann darüber hinaus weiter unterteilt werden. [Gens06] nimmt hierfür beispielsweise eine Unterscheidung in Primär- und Sekundärkontext vor. Der Primärkontext umfasst Eigenschaften, die ausreichend sind, um eine aktuelle Situation eindeutig zu beschreiben. Der Sekundärkontext liefert dagegen zusätzliche Informationen, die den Kontext zwar genauer

beschreiben, jedoch auf dem Primärkontext aufbauen. Die Schwierigkeit besteht darin, zu ermitteln, welche Informationen das sind. Dazu ist zunächst zu spezifizieren, wessen Kontext gemeint ist. In [DeSe03] werden hierzu zwei Sichtweisen für Netzwerke unterschieden. Dabei handelt es sich zum einen um den Kontext des Netzwerkes, d. h. für die kontextsensitiven Anwendungen sind die Positionen der Netzknoten, die Verkehrslast, Netzübergänge, Handover usw. von Bedeutung. Zum anderen besteht die Möglichkeit, dass die Anwendung auf den Kontext des Nutzers angewiesen ist. Ergänzend dazu wird in [DeLS05] auch noch der Kontext des Endgerätes eingeführt. Damit werden alle Kommunikationsinstanzen eines Netzwerkes umfasst.

Im Rahmen dieser Arbeit wird der Begriff Kontext vorrangig auf den Nutzer bezogen. Es ist aber durchaus auch möglich, den Kontext von Netzwerk und Endgerät mit zu berücksichtigen.

Um den Kontext abbilden zu können, müssen dessen Eigenschaften beschrieben werden. Der folgende Abschnitt zeigt, wie dies mit Hilfe von Kontexttypen unterstützt wird.

4.3 Kontexttypen

Je mehr Informationen über den Kontext bekannt sind, desto besser kann dieser beschrieben bzw. abgebildet werden. Würde also versucht werden, aus den zur Verfügung stehenden Informationen einen bestimmten Kontext zu beschreiben, nähme die Genauigkeit dieser Beschreibung mit der Menge der Informationen zu. Eine vollständige Übereinstimmung zwischen beiden scheitert aber spätestens an der Heisenbergschen Unschärferelation. Daraus wird ersichtlich, dass es unmöglich ist, den Kontext eines Nutzers vollständig abzubilden. Eine genaue Kopie bzw. Reproduktion einer Entität kann also niemals erfolgen. Andererseits besteht dazu aber auch keine Notwendigkeit, da die kontextsensitiven Dienste in der Regel nur eine begrenzte Anzahl von Informationen für eine Anpassung benötigen. Für einen einzelnen kontextsensitiven Dienst sind somit nur ganz bestimmte Informationen relevant, die systematisch durch Kontexttyp und Kontextinformation beschrieben werden:

Definition 4.2 (Kontexttyp) *Als Kontexttyp wird ein charakteristischer Bestandteil eines Kontextes bezeichnet, der durch seinen Informationsgehalt zur Beschreibung dieses Kontextes beiträgt. Ein Kontexttyp kann dabei in weitere untergeordnete Kontexttypen gegliedert sein, die dann die komplette Information beinhalten.*

Definition 4.3 (Kontextinformation) *Die Kontextinformation stellt eine Größe dar, mit der der Kontexttyp charakterisiert wird. Dabei kann ein Kontexttyp durch mehrere Kontextinformationen beschrieben werden.*

Abbildung 4.1 zeigt noch einmal den Zusammenhang zwischen Kontext, Kontexttyp und Kontextinformation. Der Kontext wird dabei durch verschiedene Kontexttypen dargestellt. Diese können durch weitere Subtypen untergliedert und dann durch Kontextinformationen beschrieben werden. Eine andere Möglichkeit besteht darin, die Kontextinformationen direkt an den Kontexttyp zu binden. Natürlich könnten auch die Subtypen noch weiter untergliedert werden. Am Ende einer solchen Hierarchie muss jedoch immer mindestens eine Kontextinformation den jeweiligen Typ beschreiben.

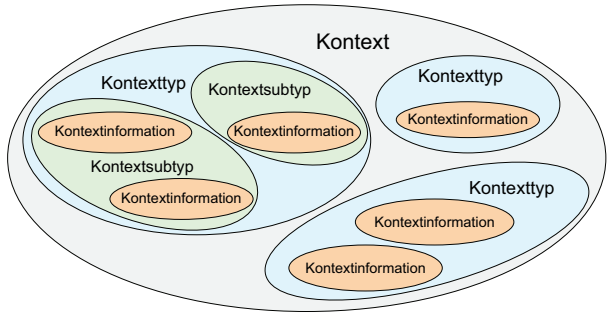


Abbildung 4.1: Einordnung von Kontext, Kontexttyp und Kontextinformation

Um die einzelnen Kontextinformationen charakterisieren zu können, wird in der Literatur häufig zuerst eine Klassifikation des Kontextes vorgenommen. Für den Nutzer unterscheidet [DeAb97] z. B. drei Kategorien des Kontextes (physisch, basierend auf Nutzeraktionen und emotional). Jeder Kategorie werden verschiedene Kontexttypen zugeordnet, die durch die jeweiligen Kontextinformation näher beschrieben werden. Der physische Kontext umfasst die Position des Nutzers, seine Orientierung, die Zeit, das Datum usw. Zum Kontext, der die Nutzeraktionen beschreibt, gehören beispielsweise aktuelle Bewegungen und Handlungen des Nutzers. Schließlich beinhaltet der emotionale Kontext den gefühlsmäßigen Zustand des Nutzers. Zu berücksichtigen ist bei dieser Klassifikation, dass sich die einzelnen Kategorien nicht nur auf den Nutzer selbst sondern auch auf andere Personen und Objekte in seiner Umgebung abbilden lassen. Eine direkte Verbindung zwischen diesen und dem Nutzer fehlt jedoch. In [LeDS05] wird dagegen zwischen persönlichem, technischem und Umgebungskontext unterschieden. Im Gegensatz zur ersten Kategorisierung beinhaltet lediglich der persönliche Kontext Informationen über den Nutzer als Person. Der technische Kontext dagegen umfasst solche Geräte, Werkzeuge, Hilfsmittel usw., die der Nutzer verwendet. Schließlich gehören zum Umgebungskontext Informationen zur Umwelt des Nutzers. Dazu gehören beispielsweise seine Position, andere Personen in der Nähe und das Wetter. Wie leicht zu erkennen ist, beziehen sich diese Kategorien direkt auf den Nutzer, schließen aber dessen Umgebung, im Gegensatz zu [DeAb97] direkt mit ein. Dadurch wird diese Klassifizierung wesentlich flexibler und kann den Kontext eines Nutzers besser beschreiben.

Die Kontexttypen können zwar einzelnen Kategorien zugeordnet werden. Deren Spektrum ist jedoch sehr vielfältig und wächst mit der Entwicklung neuer kontextsensitiver Dienste immer weiter. Allerdings wird noch einmal darauf hingewiesen, dass die Verwendung der Kontexttypen auf den jeweiligen Anwendungsfall bzw. Dienst zugeschnitten sein sollte. Tabelle 4.1 zeigt dazu ein Beispiel, dass sich an [LeDS05, Lewa07] anlehnt. Dort ist ein Datenbankeintrag dargestellt, der verschiedene Kontexte, Kontexttypen und die zugehörigen Informationen enthält. Die Kontexttypen sind hierbei untergliedert (siehe Spalten 2 bis 4). Es werden einige repräsentative Vertreter aufgelistet und beschrieben. Bei der Auswahl werden auch zugehörige mögliche Kontextinformationen angegeben, um zu zeigen, wie spezifisch solche Informationen dargestellt werden können. Zu beachten ist auch, dass die Kon-

texttypen in einigen Fällen auch noch weiter untergliedert werden, was gerade für die Verwaltung in Datenbanken von Vorteil sein kann.

Kontext	Kontexttyp	Kontextsubtyp	Kontextsubsubtyp	Kontextinformation
PERSÖNLICHER KONTEXT	**BENUTZERDATEN**	PERSONALANGABEN	ID	***0176***
			Name	***Anna***
		INTERESSE	Hobbys	***Musik***
			aktuelle Interessen	***Einkaufen***
	GRUPPEMITGLIED-SCHAFT	ZUGEHÖRIGKEIT	Einzelperson	***x***
			Gruppe	***---***
	STATUS	KÖRPERFUNKTIONEN	Herzfrequenz (Schläge/Minute)	***70***
			Blutdruck (mmHg)	***120/80***
			Körpertemperatur (°C)	***36,2***
			Zuckerspiegel (mg/dl)	***---***
		BEWEGUNGS-GESCHWINDIGKEIT	Langsam	***---***
			Normal	***x***
			Schnell	***---***
	BEHINDERUNGEN/ EINSCHRÄNKUNGEN		Art	***Sehbehinderung***
			Grad (%)	***50***
TECHNISCHER KONTEXT	**GERÄT**	TYP		***Pocket PC***
		MOBILITÄT	Stationär	***---***
			Mobil	***x***
		DISPLAY	Auflösung (dpi)	***240x320***
			Farbe	***Monochrom***
		HARDWARETASTEN	Anzahl	***4***
			Notruf	***x***
		AUDIOAUSGANG		***x***
		SPRACHSYNTHESE		***x***
		BATTERIESTATUS (%)	Ladezustand (%)	***30***
			Restlaufdauer (Minuten)	***35***
	TECHNISCHE UMWELT	NETZ	ISDN	***---***
			GSM/ GPRS	***x***
			WLAN	***---***
UMGEBUNGSKONTEXT	**ZEIT**	ART	verfügbare Zeit (Stunden)	***2,5***
		TYP	Datum	***15.12.2004***
			Uhrzeit	***08:30:46***
	POSITION	ORT	außerhalb von Gebäuden	***x***
			innerhalb von Gebäuden	***---***
		ART	GPS	***10°56´22´´öL 50°40´57´´nB***
			GSM	***---***
			RFID-Daten	***---***
	ORIENTIERUNG	ORIENTIERUNGSART	Richtung	***Nord***
			Koordinaten	***---***
	UMGEBUNG	WETTERBEDINGUNGEN	Temperatur (°C)	***10***
			Niederschlag	***Regen***

x für vorhanden; **---** für nicht vorhanden

Tabelle 4.1: Datenbankeintrag zur Kontextbeschreibung nach [LeDS05, Lewa07]

Die bis heute entwickelten Systeme und Dienste belegen, dass bei der Vielzahl möglicher Kontexttypen zwei von essentieller Bedeutung sind. Zum einen ist dies die Zeit und zum anderen die Position des Nutzers.

Die Zeit ist deshalb wichtig, weil der Kontext (von einem Nutzer) unmittelbar von der Zeit abhängt. D. h., ein Nutzer besitzt zu unterschiedlichen Zeitpunkten verschiedene Kontexte. Inwieweit sich das auf die Kontextbeschreibung auswirkt, hängt davon ab, welche Kontexttypen verwendet bzw. ausgewertet werden. Das auftretende Problem der Alterung der Kontextinformationen und der damit in Zusammenhang stehenden Aktualität und Gültigkeit wirkt sich auch auf die Diensterbringung aus. Dem Nutzer kann nur dann ein optimal auf ihn angepasster Dienst angeboten werden, wenn die zu ihm korrelierenden Kontextinformationen gültig sind. Dazu wird auch in [Nies02] richtig festgestellt, dass die Häufigkeit der Aktualisierung von Kontextinformationen an das dynamische Verhalten des Kontextes angepasst werden muss.

Die Position stellt dagegen eine elementare Information des Umgebungskontextes dar. Das ist gerade für kommerzielle Dienstleister interressant, die ihre Angebote dann dem Nutzer unterbreiten können, wenn sich dieser in der Nähe des Dienstleisters befindet (z. B. für Werbung) oder aber den Dienst tatsächlich benötigt (z. B. für eine ortsgebundene Informationsbereitstellung in einem Museum). Wenn bei der Diensterbringung nur die Position des Nutzers eine Rolle spielt, werden diese auch als *Location Based Services* (LBS) bezeichnet. Dabei ist es vom Dienst abhängig, wie genau eine Positionsbestimmung erfolgen muss.

Ebenso wichtig ist die korrekte Auswahl der benötigten Kontexttypen. Natürlich kann eine Gesamtübersicht über Kontexttypen niemals vollständig sein, da sich auch die kontextsensitiven Anwendungen immer weiterentwickeln werden. D. h., Informationen, die heute noch vollkommen irrelevant sind, könnten in Zukunft zur unabdingbaren Voraussetzung für bestimmte Anwendungen werden. Dessen ungeachtet sollten immer nur die wirklich benötigten Kontexttypen in den Prozess der Diensterbringung einfließen. Irrelevante Informationen erhöhen den Overhead und können sich damit negativ auf die Performance des gesamten Systems auswirken. Dem kann durch optimierte Prozesse bei der Kontexterfassung und -verarbeitung entgegengewirkt werden. Welche Einflüsse hierbei eine Rolle spielen, zeigt der folgende Abschnitt.

4.4 Kontexterfassung und -verarbeitung

Bevor Kontextinformationen von einer Anwendung oder einem Dienst genutzt werden können, müssen diese erfasst und übertragen werden. Die Erfassung von Kontextinfomationen stellt damit die Basis eines jeden kontextsensitiven Systems dar. Häufig ist zusätzlich auch noch eine Anpassung des Datenformats an den Dienst notwendig, der die Kontextinformation nutzt. Die Methoden zur Erfassung von Kontextinformationen sind so vielfältig wie die Kontexttypen selbst. In [LeDS05] wurde diese Thematik ausführlich vorgestellt. Ein Ergebnis daraus ist eine Klassifizierung zur Art und Weise, wie diese Informationen erfasst werden können. Es werden explizites (*explicit*), implizites (*implicit*) und sensorisches (*sensory*) Feedback unterschieden. Diese drei Methoden ergänzen sich und können in Kombination zu einer sehr guten Beschreibung des Kontextes führen.

Beim expliziten Feedback wird der Nutzer, dessen Kontext erfasst werden soll, direkt befragt. Das kann über einen Fragebogen, eine interaktive Eingabe o. ä. erfolgen. Diese Methode kann sehr genaue Informationen liefern. Die Befragung nimmt jedoch auch Zeit des Nutzers in Anspruch und kann ihn ggf. bei seiner aktuellen Tätigkeit stören. Hinzu kommt, dass der Nutzer nur wenig oder keine Motivation besitzt, Fragen zu beantworten. Die Gefahr des Demotiviertseins steigt dabei mit der Zahl der Fragen. Außerdem ist der Nutzer hier als Fehlerquelle zu betrachten. So kann er beispielsweise Fragen falsch interpretieren, was sich negativ auf das Ergebnis auswirken würde. Deshalb ist es wichtig, dass solche Fragen einfach und eindeutig gestellt werden. Die erwarteten Informationen sollten dabei möglichst zeitinvariant sein, um wiederholtes Fragen zu vermeiden.

Beim impliziten Feedback wird versucht Kontextinformationen zu sammeln, indem der Nutzer und dessen Interaktionen mit seinen Geräten beobachtet werden. Ruft ein Nutzer beispielsweise eine bestimmte Information mit seinem Gerät ab, registriert

dies das System, klassifiziert die Anfrage und kann so einzelne Interessen oder Bedürfnisse des Nutzers erkennen. Mit diesem Wissen können dem Nutzer dann andere ihn (vielleicht) interessierende Informationen bzw. Dienste angeboten werden. Von den Prozessen der Kontexterfassung bemerkt der Nutzer nichts und wird dadurch auch nicht belästigt. Andererseits ist eine gewisse Anzahl von Informationen nötig, bevor korrekte Aussagen über dessen Bedürfnisse oder Interessen möglich sind.

Das sensorische Feedback erlangt mit fortschreitender Technik immer größere Bedeutung. Hierbei werden die Kontextinformationen mit Hilfe von Sensoren ermittelt. Diese können sich beispielsweise in der Umgebung des Nutzer, in dessen Geräten, seiner Kleidung oder als Implantat in seinem Körper befinden. Es können bestimmte physikalische Werte aufgenommen und an das Kontextsystem geliefert werden. Der Begriff Sensor kann dabei sehr weit gefasst sein. Dazu gehören einfache Schaltungen (z. B. Temperaturfühler) genauso wie komplexe Geräte (z. B. Mobilfunktelefone zur Positionsbestimmung, Kameras). Hinzu kommt auch die Klasse der Labels, die zur Markierung von Objekten dienen. Darunter zählen Barcodes, *Radio Frequency Identification Tags* (RFID-Tags) usw. Es werden passive und aktive Labels unterschieden. Im Gegensatz zur passiven Variante besitzen aktive Labels eine eigene Energiequelle. Diese Energiequelle ermöglicht z. B. bei einem aktiven RFID-Tag das Versenden der bei ihm gespeicherten Informationen. Das passive RFID-Tag nutzt dagegen die Energie der von einem potentiellen Interessenten gesendeten elektromagnetischen Welle, um seine Informationen zu versenden. Bei der Übertragung sind deshalb die Reichweiten von passiven RFID-Tags geringer als bei aktiven Komponenten. Als Bedingung für den Einsatz von Labels wird in [Böri02] ausgeführt, dass diese in einem bestimmten Bereich eindeutig identifizierbar sein müssen. Die Größe ist dabei von der Verwendung der Daten abhängig. RFID-Tags übertragen ihre Informationen nur auf eine Anfrage hin. Komponenten, die dagegen stetig Informationen an die Umgebung senden, werden auch hier als Leuchtfeuer (Beacons) bezeichnet. Damit können eindeutige Identifikatoren (z. B. zur Positionsbestimmung) oder auch größere Informationsmengen (z. B. zu Sehenswürdigkeiten an diesem Standort) verbreitet werden.

Neben der Erfassung der Kontextinformationen ist natürlich auch der dafür gewählte Zeitpunkt ein wesentlicher Faktor. Dieser kann wiederum als Kontexttyp betrachtet werden. Viele der Kontextinformationen sind nur für den Zeitpunkt ihrer Erfassung gültig bzw. werden, wie in Abschnitt 4.3 beschrieben, mit zunehmendem Alter ungenauer. Um dann den aktuellen Kontext des Nutzers trotzdem beschreiben zu können, ist eine wiederholte Abfrage dieser Informationen nötig. Für die zeitlichen Abstände dieser Wiederholungen ist ein Optimum zu finden. Ist die Anzahl der Wiederholungen zu hoch, sind die Informationen für die kontextsensitive Diensterbringung irrelevant und wirken sich negativ auf die Performance des Systems bzw. auf die Verkehrslast des Netzwerkes aus. Die Häufigkeit der Erfassung und Übermittlung einzelner Kontextinformationen hängt wiederum vom Kontexttyp ab. Das kann einmalig (z. B. Art und Grad der Behinderung eines Nutzers), periodisch (z. B. Position bei mobilen Nutzern) oder ereignisbasiert (z. B. Wetterinformationen, Herzfrequenz des Nutzers) erfolgen.

Wie in Abschnitt 4.3 bereits dargelegt, spielt neben der Zeit als bedeutendem Kontexttyp auch die Position für viele kontextsensitiven Dienste eine herausragende Rolle. Die Anforderungen des jeweiligen Dienstes bestimmen dabei die benötigte

Genauigkeit der Positionsbestimmung. Aktuell eingesetzte Verfahren im freien Feld nutzen beispielsweise verschiedene Varianten von GPS aber auch Mobilfunksysteme. So sind mit GPS in Verbindung mit EGNOS (*European Geostationary Navigation Overlay System*) Genauigkeiten von weniger als 4 Meter [Pott04] möglich. In Kombination mit Referenzpunkten wie beispielsweise RFIDs können die Werte noch wesentlich verbessert werden. Weitere Methoden zur Erfassung der Position im freien Feld wurden im Rahmen von Forschungen zum Einsatz für kontextsensitive Dienste in [Pott03, Heer05, Gens06] untersucht und hinsichtlich ihrer Genauigkeit und Verfügbarkeit bewertet. Die Positionsbestimmung in Gebäuden erfolgt häufig mit Hilfe fest installierter Systeme. Genutzt werden hierfür beispielsweise, wie in [Wint03] beschrieben, Infrarot, Funkbaken oder Ultraschall. Aber auch netzwerkgestützte Systeme können eingesetzt werden. In [DeSW04] wird dazu z. B. WLAN verwendet.

Die Übertragung der Kontextinformationen kann über die herkömmlichen Netzwerktechniken erfolgen, wobei ggf. bei großen Datenmengen die dort unterstützten Bitraten berücksichtigt werden müssen. Zur Beschreibung und Übertragung des Kontextes bzw. der Kontextinformationen sollte dann ein allgemein akzeptiertes Format genutzt werden. In [DeLS05] wird dafür z. B. XML (*Extensible Markup Language*) vorgeschlagen. Diese Beschreibungssprache wurde für die Übertragung von strukturierten Daten entwickelt und ist plattformunabhängig.

Neben der Erfassung und Übertragung von Kontextinformationen ist gerade bei Sensordaten auch eine Anpassung an die jeweilige Anwendung oder den Dienst notwendig. So liefern z. B. viele GPS-Empfänger lediglich Daten nach dem Standard NMEA0183 [Pott03]. Diese müssen noch entsprechend aufbereitet werden, damit eine Angabe der Position des Nutzers bzw. seines Gerätes in Breiten- und Längengrad möglich ist. Zur Navigation mit Hilfe einer Karte müssen diese Daten dann wiederum dafür angepasst werden. Durch eine Filterung und Wichtung der Kontextinformationen werden nur die tatsächlich für einen Dienst nötigen Informationen verarbeitet. Außerdem kann dadurch die Reihenfolge bei der Verarbeitung entsprechend ihrer Bedeutung erfolgen. Wurden die Kontextinformationen erfasst und übertragen, kann der Kontext abgebildet und dem Nutzer ein optimal auf ihn angepasster Dienst angeboten werden. Dazu wird im folgenden Abschnitt diskutiert, wie der Dienstbegriff im Rahmen dieser Arbeit einzuordnen ist.

4.5 Dienstbegriff

Der Begriff Dienst wird im Allgemeinen verkürzt für Dienstleistung verwendet. Dabei handelt es sich im volkswirtschaftlichen Sinne nach [DefD07] um „. . . eine von einer natürlichen Person oder einer juristischen Person zu einem Zeitpunkt oder in einem Zeitrahmen erbrachte Leistung zur Befriedigung eines Bedürfnisses“. Um den Begriff auf technische Belange abzubilden, wurde die Definition für die vorliegende Arbeit verallgemeinert:

Definition 4.4 (Dienst) *„Ein Dienst ist eine von einer Entität zu einem Zeitpunkt oder in einem Zeitrahmen erbrachte Leistung zur Befriedigung eines Bedürfnisses einer anderen Entität“.*

Nach Nutzung und Einsatzgebiet wird heutzutage zwischen einer Vielzahl von Diensten unterschieden. In der Telekommunikation finden sich beispielsweise Übermittlungs- und Teledienste, welche wiederum in Basis-, Zusatz- und/oder Mehrwertdienste gegliedert werden können. Übermittlungsdienste umfassen dabei die Funktionen des Netzwerkes, d. h. die Schichten 1–3 des OSI-Referenzmodells, während es beim Teledienst die Schichten 1–7 sind. Nach [SDHT07] beschreibt der Teledienst die Funktionen des Endgerätes. Er wird auf Basis von Übermittlungsdiensten erbracht und bietet dem Nutzer eine Schnittstelle. In den Schichtenmodellen für Kommunikationssysteme wird der Dienst als eine Menge von Funktionen einer Schicht beschrieben, die der ihr übergeordneten Schicht angeboten wird. Bezüglich der kontextsensitiven Dienste, wie sie in dieser Arbeit verwendet werden, wären dies Dienste, die in der Anwendungsschicht angeboten werden. Das können zum einen Anwendungen sein, die auf dem Endgerät des Nutzers arbeiten. Zum anderen sind solche Dienste gemeint, die in irgendeiner Form einem Nutzer zur Verfügung gestellt werden. Eine klassische Einteilung in Übermittlungs- und Teledienste ist damit nur in Grenzen möglich und muss im Einzelnen abgeklärt werden. In der Regel werden die Dienste aber von Dienstleistern angeboten, die sich in einer räumlichen Distanz zum Nutzer befinden. Die Diensterbringung erfolgt dann nach dem klassischen Client-Server-Prinzip. Mit der Entwicklung des Web 2.0 beginnt sich jedoch die Grenze zwischen Dienstleister und Nutzer aufzulösen. Somit ist es möglich, dass ein Nutzer durchaus auch Dienstleister für andere Nutzer sein kann.

Dienste lassen sich vielfältig klassifizieren. Im Folgenden werden aber nur die wichtigsten Dienstarten berücksichtigt, um das Spektrum der für die vorliegende Arbeit interessierenden Dienste aufzuzeigen. Unterschieden werden z. B. Dienste, die im *Pull*- oder *Push*-Verfahren arbeiten. Bei der ersten Variante muss der Nutzer eine Anfrage stellen, um auf einen Dienst zuzugreifen (z. B. Telefonauskunft). Dagegen wird beim *Push*-Verfahren der Dienst ohne eine explizite Anfrage des Nutzers bereitgestellt (z. B. Fernsehen). Darüber hinaus werden Dienste unterschieden, mit denen der Nutzer direkt interagiert (z. B. Informationsdienste), als auch solche, bei denen der Nutzer nur passiv Einfluss ausübt. Letzteres betrifft z. B. Dienste der Hausautomation. Dort könnte die Lichtsteuerung basierend auf der Position des Nutzers erfolgen. [Gens06] beschreibt diese Dienste als reaktiv und proaktiv. Darüber hinaus muss in Anlehnung an Kapitel 2 berücksichtigt werden, dass Dienste heutzutage durchaus auch mobil sein können. Das stellt natürlich erhöhte Anforderungen an die Dienstsuche und anschließende Kommunikation. Schließlich ist noch zu erwähnen, dass davon ausgegangen wird, dass die Dienstanfrage mindestens elektronisch kommuniziert wird. Die Diensterbringung selbst muss dagegen nicht elektronisch erfolgen.

Der bis hierhin definierte Dienst wird noch um das Attribut kontextsensitiv erweitert. In Verbindung mit den Abschnitten 4.2 und 4.3 bedeutet dies, dass der Dienst mit Hilfe von Kontextinformationen optimal an die Bedürfnisse des Nutzers angepasst und diesem bereitgestellt werden kann. Welche Kontexttypen benötigt werden, bestimmt der jeweilige Dienst selbst. Hierbei wird zwischen obligatorischen und optionalen Kontexttypen unterschieden. Die Anzahl unterliegt keiner Beschränkung und kann bei einzelnen Diensten abhängig vom Nutzer schwanken. Diese Annahme führt zu einer interessanten Schlussfolgerung. Dazu wird ein kontextsensitiver Dienst mit einem herkömmlichen (nicht kontextsensitiven) Dienst verglichen. Hierbei könnte es sich beispielsweise um Informationsdienste handeln, die auf die gleiche

Datenbasis zugreifen. Unterscheidet dann der kontextsensitive Dienst zur Ermittlung des Kontextes nur, ob und mit welchem Grad ein Nutzer behindert ist, liefert er das gleiche und optimale Ergebnis wie ein herkömmlicher Dienst, wenn der anfragende Nutzer nicht behindert war. In diesem Spezialfall wäre keine Übermittlung und Auswertung des Behinderungsgrades nötig. Damit bildet der herkömmliche Dienst eine Untermenge des kontextsensitiven Dienstes. Dem gegenüber kann der Fall eintreten, dass ein kontextsensitiver Dienst bestimmte Kontextinformationen unbedingt benötigt, um ein Ergebnis liefern zu können. Wenn diese Informationen aber nicht zur Verfügung stehen, muss im Einzelnen entschieden werden, ob dem Nutzer ein alternativer Dienst angeboten werden kann. Das Ergebnis ist dann zwar nicht optimal, aber dafür möglicherweise noch nutzbar. Sofern keine Alternative besteht, ist die Dienstanfrage negativ zu beantworten.

Damit sich kontextsensitive Dienstangebote flächendeckend durchsetzen können, spielen neben der allgemeinen Benutzerfreundlichkeit der Geräte und Anwendungen auch Datenschutz und Sicherheit eine wichtige Rolle. Schon in [Diet05] wurde für LBS richtig festgestellt, dass die Nutzer nicht bereit sind, ständig und überall mit Werbung konfrontiert zu werden. Außerdem besteht mit solchen Diensten die Möglichkeit den Nutzer zu überwachen und ohne sein Wissen Profile anzulegen. Kontextsensitive Dienste können dabei durch die enorme Anzahl von möglichen Kontextinformationen eine noch bessere Plattform bieten. Für einen Erfolg solcher Dienste müssen deshalb deren Anbieter dem Nutzer den größtmöglichen Schutz bieten.

Die Anzahl der bereits in konventionellen Netzen vorhandenen kontextsensitiven Dienste steigt im Zuge der Personalisierung ständig an. Zu den bekanntesten Vertretern gehören derzeit Informations- und Benachrichtigungsdienste, Dienste zum Lokalisieren, Touristenführer und Gedächtnishilfen. Zur Realisierung innerhalb von Kommunikationsnetzen werden Architekturen benötigt, die einen einfachen und schnellen Zugriff auf die Dienste erlauben. Der folgende Abschnitt beschreibt diesbezüglich den aktuellen Stand.

4.6 Aktuelle Methoden zur Dienstbereitstellung

Um Dienste bereitstellen zu können, bedarf es entsprechender Architekturen, die sowohl die Dienstsuche und -auswahl als auch die Kommunikation zwischen Diensterbringer und Dienstleister unterstützen. Grundlage für eine Diensterbringung bildet die zu Beginn stattfindende Suche nach dem Dienst. Die dafür notwendigen Kommunikationsprozesse werden durch entsprechende Protokolle unterstützt. Diese Protokolle und Architekturen werden in diesem Abschnitt hinsichtlich ihrer Kommunikations- und Netzwerkeigenschaften gegenübergestellt. Dabei kann jedoch nur ein Überblick über die wichtigsten Techniken gegeben werden. Anhand dieser wird aufgezeigt, wo die grundsätzlichen Mängel heutiger Dienstarchitekturen liegen.

4.6.1 Art der Dienstsuche

Die einzelnen Dienstarchitekturen lassen sich nach [Renh07b] prinzipiell in zwei Gruppen gliedern. Dies erfolgt über die Art der Verwaltung der Dienste. Damit ein Nutzer einen Dienst abrufen kann, muss er dessen Adresse bzw. die des zugehörigen Servers kennen. Die diesbezüglichen Informationen müssen im Netz bereitgestellt werden. Eine Möglichkeit dafür zeigt Abbildung 4.2. Dort wird ein zentraler

Verzeichnisdienst verwendet, bei dem alle Diensterbringer oder Server und deren Adressen aufgelistet sind. Die Server müssen dazu ihre Dienste registrieren (1). Gefunden werden die Verzeichnisdienste in der Regel über ein Broad- oder Multicast. Die Clients können ihre Anfragen (2) jetzt an diesen Verzeichnisdienst stellen. Sie erhalten dann von dort die entsprechenden Informationen (3). Daraufhin kann der Client direkt mit dem Server kommunizieren und auf den gewünschten Dienst zugreifen (4). Diese Art der Dienstsuche wird häufig in Infrastrukturnetzen angewendet. Mit dem zentralen Verzeichnisdienst ist eine Lastverteilung bezogen auf die Server gut skalierbar.

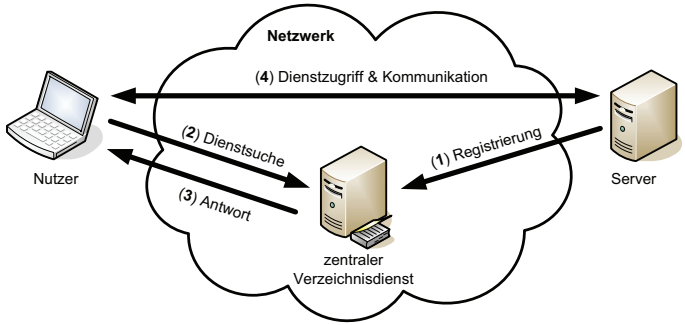


Abbildung 4.2: Dienstsuche auf Basis eines zentralen Verzeichnisdienstes

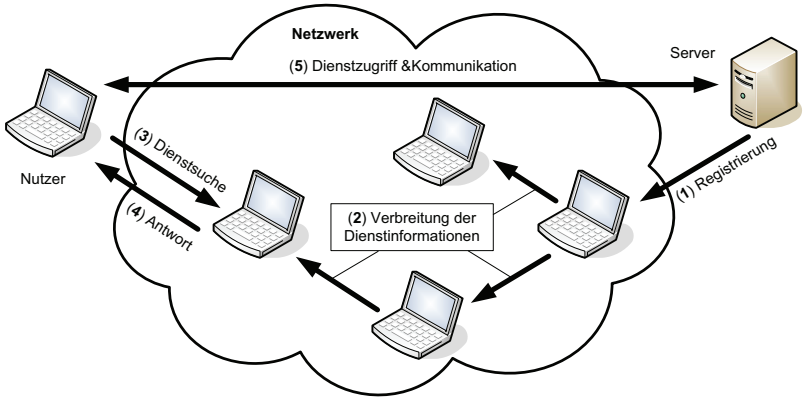


Abbildung 4.3: Dienstsuche ohne zentralen Verzeichnisdienst

Wird bei Ad-hoc-Netzen die dynamische Topologie berücksichtigt, erscheinen zentrale Verzeichnisdienste als ungeeignet. Deshalb werden hier in der Regel verzeichnislose Architekturen wie in Abbildung 4.3 favorisiert. Zur Verbreitung der Informationen über Dienstangebote (1) und zur anschließenden Suche nach Diensten werden wiederum Broad- und Multicasts genutzt. Diese Informationen werden je nach Architektur bzw. Protokoll durch zwei Speicherstrategien im Netzwerk vorgehalten. Zum

einen können, wie in Abbildung 4.3 durch (2) dargestellt, alle Knoten empfangene Dienstinformationen zwischenspeichern. Zum anderen können diese aber auch entsprechend bestimmter Algorithmen im Netzwerk verteilt werden. Ein Client sendet seine Anfrage (3) einfach in das Netzwerk. Diese wird dann bis zu einem Knoten weitergeleitet, der darauf antworten kann (4). Mit den erhaltenen Adressinformationen können nun Client und Server wieder direkt miteinander kommunizieren (5). Trotz der flexibleren Gestaltung kann in diesen Systemen nicht garantiert werden, dass ein Dienst tatsächlich gefunden wird. Die Wahrscheinlichkeit für eine erfolgreiche Dienstsuche ist aber wesentlich höher, als bei Architekturen mit einem zentralen Verzeichnisdienst.

Natürlich besteht auch prinzipiell die Möglichkeit beide Modi zu kombinieren. So unterstützt beispielsweise das *Service Location Protocol* (SLP) [GPVD99] beide Betriebsarten.

4.6.2 Architekturen und Protokolle zur Dienstsuche

Entsprechend der Vielzahl von Diensten existiert auch eine große Anzahl an verschiedenen Dienstarchitekturen und -protokollen. Viele davon nutzen zur Dienstsuche sogenannten Attribute. Diese dienen einer genaueren Dienstbeschreibung. Damit kann beispielsweise ein Dienst „Drucker“ mit dem Attribut „Laser“ näher bestimmt werden. Dies kann einerseits auch als Kontexttyp interpretiert werden. Andererseits ist eine solche Dienstbeschreibung statisch. D. h., die sich aus den Attributen ergebenden Kontextinformationen ändern sich nicht mit dem Kontext des Nutzers. Eine kontextsensitive Diensterbringung ist somit nur sehr begrenzt möglich.

Viele der Architekturen und Protokolle zur Dienstsuche sind für bestimmte Anwendungen und Einsatzzwecke entwickelt worden. [Renh07b] unterscheidet beispielsweise drei Entwicklergruppen (Forschung, Softwarehäuser und Industriestandards), deren unterschiedlichen Anforderungen und Bedürfnisse in den jeweiligen Spezifikationen berücksichtigt wurden. Deshalb sind diese Verfahren auch häufig inkompatibel zueinander. Viele wissenschaftliche Arbeiten beschäftigen sich mit dieser Thematik, da sie durch die stetige Weiterentwicklung der Dienste nie an Attraktivität verliert. Daraus ergibt sich notwendigerweise auch, dass beispielsweise verschiedene frühere Ansätze aufgrund neuer Netzwerktechnologien mittlerweile veraltet sind oder sich bis heute nicht durchsetzen konnten. In Anlehnung an die Übersichten in [Wang02, Edwa06, Renh07b] zeigt Tabelle 4.2 auf Seite 46 verschiedene Eigenschaften einer repräsentativen Auswahl aktueller Dienstarchitekturen und -protokolle, in die im Folgenden kurz eingeführt werden soll.

AODV-SD (AODV-*Service Discovery*[1])[ToGM04] ist ein für Ad-hoc-Netze entwickeltes Protokoll zur Dienstsuche. Dabei handelt es sich um eine Erweiterung des reaktiven AODV, basierend auf den Vorschlägen in [KoPe02]. Die dort verwendeten Pakete zur Routensuche enthalten auch die Dienstanfrage des Clients und die zugehörigen Dienstattribute. Das entsprechende Antwortpaket liefert dann den *Uniform Resource Locator* (URL) des gesuchten Dienstes zurück. Die Suche nach Diensten ist auf das Ad-hoc-Netz beschränkt. Für den Zugriff der Anwendungen auf die Architektur dient ein spezielles *Application Programming Interface* (API).

[1] Im Original bedeutet AODV-SD „AODV with extensions for service discovery“.

Bonjour ist eine von der Apple Inc. entwickelte Architektur für IP-Netzwerke, welche auf dem *Domain Name System* (DNS) basiert. Die Funktionen werden dabei durch spezielle Erweiterungen ermöglicht. Die Dienstsuche erfolgt mit dem *DNS-Based Service Discovery* (DNS-SD) [ChKr06a]. Das *multicast DNS* (mDNS) [ChKr06b] unterstützt die Nutzung von Bonjour, wenn kein DNS-Server zur Verfügung steht.

DEAPspace [HHMN+01] ist ein von der *IBM Research Devision* entwickeltes Framework und arbeitet als verteiltes System. Dabei wurden speziell die Erfordernisse für Ad-hoc-Netze berücksichtigt. Jeder Knoten besitzt eine Liste mit Diensten und nutzt Broadcasts, um die eigenen Dienste bekannt zu machen. Da es sich um ein *Push*-System handelt, lassen sich damit auch relativ einfach LBS realisieren. Zwar ist DEAPspace für Ad-hoc-Netze entwickelt worden, jedoch ist nach [HHMN+01] auch ein Einsatz in Infrastrukturnetzen möglich. Nach [Renh07b] dienen die unterstützten Attribute zur detaillierten Dienstbeschreibung. Allerdings erfolgt mit DEAPspace keine explizite Dienstsuche.

Das **GSD** (*Group-based Service Discovery Protocol*) [CJFY02] ist ebenfalls speziell für Ad-hoc-Netze entwickelt worden. Bei diesem Protokoll zeigt ein Knoten seinem Nachbarn die von ihm angebotenen Dienste mittels *Advertisements* an. Das Besondere an GSD ist, dass die Dienstbeschreibung mit Hilfe von *Darpa Agent Markup Language* und *Ontology Inference Layer* (DAML+OIL) erfolgt. Dadurch können Dienste auch in Gruppen, Klassen sowie Subklassen gegliedert und abgebildet werden. Jeder Knoten eines Ad-hoc-Netzes besitzt Informationen über das Dienstangebot seiner Nachbarn. Diese Dienste werden gruppiert und die Gruppeninformationen beim Verbreiten des eigenen Dienstangebotes mit übertragen. Die Bekanntmachung der Dienste erfolgt in periodischen Abständen. Darüber hinaus ist auch eine Dienstsuche via Broadcast möglich. Attribute werden dabei zwar nicht unterstützt, eine beliebig genaue Spezifikation ist aber über die Gruppierung möglich. Für Infrastrukturnetze ist diese Methode der Dienstverbreitung jedoch ineffizient. Dort ist innerhalb eines Subnetzes keine Gruppierung nötig, weil alle Knoten direkt miteinander kommunizieren.

Jini [Sun 01] ist eine von Sun entwickelte Architektur und basiert auf Java. Der Dienstzugriff erfolgt mit Hilfe der *Remote Method Invocation* (RMI). Dazu werden die Dienste bei einem *Lookup Service* (LUS) registriert, indem dort entsprechend Java-Objekte hinterlegt werden. Der LUS ist wahlweise über Multicast oder Unicast erreichbar. Er kann auch Informationen zu anderen LUS in „fremden“ Netzen vorhalten. Knoten, die einen Dienst nutzen möchten, laden den Code vom LUS und greifen damit auf den Dienst zu. Während [Wang02] Jini auch in mobilen Umgebungen für geeignet hält, erklärt [ToGM04], dass es nicht für Ad-hoc-Netze geeignet ist. Jini unterstützt auch die Verwendung von Attributen.

LSD (*Lightweight Service Discovery*) [LiLa05] ist ebenfalls für Ad-hoc-Netze entwickelt worden. Es basiert auf dem proaktiven OLSR und unterliegt deshalb den gleichen netzwerktechnischen Einschränkungen wie AODV-SD. Das Besondere an LSD ist, dass es sowohl mit Verzeichnissen als auch verzeichnislos arbeiten kann. Um eine Dienstanfrage zu starten, wird auch hier die Möglichkeit der Paketerweiterung von OLSR genutzt. Darüber erfolgen das Anzeigen der Dienste im Netz,

Dienstanfragen, -antworten und die Registrierung der Dienste im Verzeichnis. Leider erläutert [LiLa05] nicht näher, ob eine Unterstützung von Attributen vorgesehen ist.

Salutation [Cons00] ist ein Protokoll des *Salutation Consortium*. Diese Vereinigung aus verschiedenen Firmen hat sich zwar 2005 aufgelöst [Renh07b], der Vollständigkeit wegen wird es aber mit aufgeführt. Außerdem dient dieses Protokoll durch seine Eigenschaften und Flexibilität auch heute noch für viele Entwickler als Maßstab. Nach [Wang02] bestand das Ziel der Entwicklung darin, eine Architektur zu entwickeln, die plattform-, betriebssystem- und netzwerkunabhängig ist. Dabei dient der sogenannte *Salutation Manager* (SM) zum Registrieren der Services, zur Dienstsuche, zum Test der Dienstverfügbarkeit und zur Kommunikationssteuerung. Mit Hilfe eines API ist die Interaktion einer Anwendung mit dem Salutation-Protokoll möglich. Auch [Pasc01] belegt, dass mit dieser Philosophie eine Zusammenarbeit mit anderen Architekturen zur Dienstsuche möglich ist. Dementsprechend werden auch Attribute unterstützt. Zusätzlich kann Salutation auf jede Netzwerktechnologie aufsetzen, was dessen Einsatzfähigkeit äußerst flexibel macht.

SDP (*Service Discovery Protocol*) [Blue04] wurde für Bluetooth entwickelt und dient dort zur Dienstsuche, welche auch mit Hilfe von Attributen erfolgen kann. Da Bluetooth, wie in Abschnitt 2.3 beschrieben, immer über einen Master kommuniziert, erfolgt auf diese Art und Weise auch die Dienstsuche. SDP setzt direkt auf dem *Logical Link Control and Adaption Protocol* (L2CAP) auf, was in der Sicherungsschicht des Bluetooth-Protokollstacks liegt. Deshalb ist gegenwärtig eine Suche nach Diensten nur über einen Hop möglich. Die Basis des autokonfigurierten Zugriffes auf Dienste bilden die so genannten Profile. Die entsprechende Spezifizierung erfolgt über die Bluetooth-SIG.

SLP (*Service Location Protocol*) wurde von der *Internet Engineering Task Force* (IETF) spezifiziert und umfasst die RFCs 2608 [GPVD99], 2609 [GuPK99], 2614 [KeGu99], 3059 [Gutt01] und 3082 [KeGo01]. Das Protokoll setzt auf IP auf und sieht drei Agenten vor. Der Nutzeragent (*User Agent*) sucht einen Dienst. Der Dienstagent (*Service Agent*) bietet diesen Dienst an und der optionale Verzeichnisagent (*Directory Agent*) stellt ein Verzeichnis bereit, bei dem die Dienste registriert und abgerufen werden können. Die zugehörige Adresse wird manuell konfiguriert oder via Multicast gesucht. Nach [Renh07b] kann sie dem Client auch mit Hilfe des *Dynamic Host Control Protocols* (DHCP) vermittelt werden. Darüber hinaus bietet SLP auch die Möglichkeit direkt nach einem Dienst zu suchen. Attribute werden unterstützt.

UPnP (*Universal Plug and Play*) [UPnP06] ist eine von der Microsoft Corporation entwickelte Architektur, die für kleinere Heimnetzwerke bestimmt ist. Es dient zur Suche und Steuerung von Geräten und Diensten in einem IP-Netzwerk. Dazu muss im Netzwerk mindestens ein Steuergerät arbeiten. Alle Geräte, die an das Netzwerk angeschlossen werden, müssen ihre Dienste über das *Simple Service Discovery Protocol* (SSDP) am Steuergerät anmelden. SSDP arbeitet hierbei mit Multicast. Des Weiteren kann das Steuergerät darüber im Netz auch nach Diensten anfragen. Außergewöhnlich dabei ist, dass UPnP keine attributbehaftete Dienstsuche unterstützt [Wang02, Edwa06].

Verfahren	Funktionalität	Verzeichnisdienst	Reichweite	Netzwerke	Flexibilität	Steuerbarkeit	Kontext
AODV-SD	Netzwerk	ohne	–	–	+	–	o
Bonjour	Anwendung	mit	–	o	+	–	–
DEAPspace	Anwendung	ohne	–	+	+	–	–[2]
GSD	Anwendung	ohne	–	o	+	–	o[2]
Jini	Anwendung	mit	+	o	o	–	o
LSD	Netzwerk	beides	–	–	+	–	–[2]
Salutation	Anwendung	mit	+	+	+	–	o
SDP	Anwendung	mit	–	–	–	–	o
SLP	Anwendung	beides	+[3]	o	+	–	o
UPnP	Anwendung	ohne	–	o	o	–	–

Tabelle 4.2: Aktuelle Verfahren zur Dienstsuche

Die in Tabelle 4.2 dargestellten Verfahren werden auf solche Eigenschaften verglichen, die essentielle Anforderungen für eine allgemeine Dienstarchitektur darstellen. Die folgende Auflistung beschreibt die Bedeutung dieser Eigenschaften.

Funktionalität: Mit Funktionalität wird beschrieben, wie die Verfahren arbeiten. Die Einteilung ist an den Protokollstack angelehnt. Ein Verfahren kann entweder Funktionen realisieren, die bis zur Anwendungsschicht reichen oder es arbeitet lediglich auf den Schichten, die Netzwerkfunktionen (incl. Routingprotokolle) anbieten.

Verzeichnisdienst: Beim Verzeichnisdienst wird eine Aussage darüber getroffen, ob bei der in Abschnitt 4.6.1 beschriebenen Art der Dienstsuche, mit zentralen Verzeichnissen gearbeitet wird. Nach [LiLa05] bieten Architekturen mit einem Verzeichnisdienst in der Regel eine bessere Skalierbarkeit bei einer großen Anzahl von Dienstangeboten im Netzwerk. Problematisch ist jedoch die Nutzung in Ad-hoc-Netzen.

Reichweite: Die Reichweite gibt an, durch welche Grenzen das jeweilige Verfahren netzwerktechnisch eingeschränkt ist. (–) bezeichnet hierbei Verfahren, die nur innerhalb einer administrativen Domäne einsetzbar sind. Mit (+) werden solche bewertet, die über Domänengrenzen hinweg arbeiten können.

Hierbei ist für IP-Infrastrukturnetze zu berücksichtigen, dass Router heutzutage in der Regel aus Sicherheitsgründen keine Broad- und Multicasts weiterleiten. Diese sind somit in ihrer Reichweite auf ein Subnetz begrenzt, was einer „administrativen Domäne“ entspricht. Ad-hoc-Netze sind in der Theorie dagegen nicht an Subnetze gebunden. Es existieren aber Lösungen, um Daten an jeden Knoten innerhalb eines solchen Netzes zu senden (siehe z. B. [PeBRD01]). In der Regel wird dafür Multicast verwendet. Damit ist ein Ad-hoc-Netz einer „administrativen Domäne“ gleichzusetzen.

Netzwerke: Diese Eigenschaft beschreibt, in welcher Netzwerkumgebung das jeweilige Verfahren arbeitet. Dabei wird unterschieden, ob nur Infrastruktur-

[2] siehe Text
[3] sofern Verzeichnisagent genutzt wird

oder nur Ad-hoc-Netze (–) unterstützt werden oder ob es für beide Varianten geeignet ist (+). Mit (o) werden solche Verfahren bewertet, die zwar für einen Netzwerktyp entwickelt wurden, deren Einsatz in bestimmten Grenzen aber auch im jeweiligen Pendant möglich wäre, ohne dass dazu Zusätze oder Modifikationen notwendig sind.

Flexibilität: Stellt ein Maß dafür dar, inwieweit die Art der Nutzung der Dienste von dem jeweiligen Verfahren abhängig ist. Dabei wird unterschieden, ob wenige fest spezifizierte (–) oder viele (o) verschiedene Dienstarten unterstützt werden können. Mit (+) werden solche Verfahren bewertet, bei denen diesbezüglich keine Einschränkungen existieren bzw. deren Funktionen zur Erbringung des Dienstes unabhängig von Transport- und Anwendungsprotokollen ist.

Steuerbarkeit: Diese Eigenschaft beschreibt die Möglichkeit für Netzprovider, den Datenverkehr während der Kommunikation bei der Dienstsuche und -bereitstellung zu überwachen und zu monitoren. So können darüber beispielsweise Abrechnungsvorgänge entsprechend dem tatsächlichen Verkehrsaufkommen erfolgen. Es gehören weiterhin Mittel dazu, mit denen der Netzbetreiber den Verkehr steuern kann. Das schließt z. B. die Verwaltung von Zugriffen auf die Server (Lastverteilung) oder die Verweigerung von Dienstangeboten ein. (–) bedeutet hierbei, dass diesbezüglich keine, und (+), dass eine Unterstützung erfolgt.

Kontext: Kontext beschreibt, inwieweit ein Verfahren den Kontext des Nutzers bei der Suche nach einem Dienst berücksichtigt. Unterschieden wird hierbei, ob keine (–) bzw. einzelne Kontexttypen (o) einfließen oder ob eine vollständige Unterstützung (unabhängig von den Kontexttypen) (+) möglich ist.

In Tabelle 4.2 ist auf Aussagen darüber verzichtet worden, wie die Kommunikation bei den einzelnen Verfahren realisiert wird (Broad-, Multi- oder Unicast) und welche Auswirkungen sich dadurch auf die Gesamtbilanz des Verkehrs ergeben. Begründet ist dies durch die komplexen Kommunikationsprozesse und die damit in Zusammenhang stehende unzureichende Möglichkeit theoretischer Aussagen. Inwieweit ein Verfahren das Netzwerk tatsächlich belastet, hängt beispielsweise nicht nur vom Overhead der Pakete, die bei einer Initialkommunikation (Dienstregistrierung oder -anfrage) ausgetauscht werden, ab. Vielmehr spielen hier die notwendige Kommunikationsprozesse, die die Funktionen der Dienstarchitektur in ihrer Gesamtheit realisieren, eine entscheidende Rolle.

Der Vergleich in Tabelle 4.2 zeigt, dass lediglich die zwei erweiterten Routingverfahren (AODV-SD und LSD) auf Netzwerkebene arbeiten, obwohl dies Vorteile bezüglich der Flexibilität gegenüber den anwendungsbezogenen Architekturen bringen würde. So müssen dort die Netzknoten lediglich die Protokolle für die Vermittlungsfunktionen auswerten können, während bei den anderen Dienstarchitekturen in der Regel mehr Aufwand nötig ist. Die Einbindung und Bereitstellung der Dienste in den Architekturen erfolgt dabei über alle Schichten bis zur Anwendungsebene. Hinzu kommt, dass bei den erweiterten Routingverfahren die Kommunikation zwischen Client und Server nach dem Auffinden des Dienstes völlig unabhängig von den Mechanismen der Dienstsuche ist. Die Routen- und Dienstsuche erfolgt parallel. Dadurch wird der Overhead gegenüber einer aufeinander folgenden Suche reduziert. Durch

die Trennung von Dienstsuche und Diensterbringung wird mehr Flexibilität geschaffen. Dies bestätigt auch [GMTo05], wo entsprechende Messungen in einem Ad-hoc-Netz durchgeführt wurden. Diese Messungen belegen, dass bei einer Dienstsuche auf Netzwerkschicht der Overhead durch Kontrolldaten geringer ist und Dienste schneller gefunden werden als bei einer Dienstsuche auf Anwendungsschicht. Damit ist ein solches Verfahren im Ad-hoc-Netz bezüglich der Bandbreitenausnutzung und dem Energieverbrauch effizienter.

Sehr deutlich wird auch der Bezug zwischen den „früheren“ Anforderungen an die einzelnen Verfahren und der unterstützten Reichweite im Netzwerk sowie der Kompatibilität zu den Netzwerken. In der Regel sind die Architekturen auf spezielle Netzwerke und Anwendungsbereiche optimiert, was ihr Einsatzspektrum darüber hinaus eingrenzt. Eine Suche nach kontextsensitiven Diensten wird entweder gar nicht oder aber nur unzureichend unterstützt. Dem Netzwerkbetreiber bietet keines der Verfahren Zugriffspunkte. Damit ist es diesem auch nicht möglich Kommunikationsprozesse z. B. durch Lastverteilung zu optimieren.

4.6.3 Spezielle Dienstarchitekturen

Wie an der in Abschnitt 4.6.2 zusammengestellten Übersicht leicht zu erkennen ist, gibt es bei den heutigen Verfahren zur Dienstsuche noch Defizite. Eine wichtige Erkenntnis besteht darin, dass eine Dienstsuche auf Netzwerkebene sinnvoll erscheint. Um ein solches Verfahren entwickeln zu können und dabei mit Rücksicht auf Kompatibilitätsfragen einen möglichst breiten Einsatz zu schaffen, müssen zusätzlich die Anforderungen anderer existierender Architekturen und Protokolle untersucht werden. Diese lassen sich in drei Kategorien gliedern, wie durch die in den folgenden Abschnitten erläuterten repräsentativen Beispiele belegt werden kann. Dazu gehören:

- Architekturen, die für Funktionen wie Dienstsuche, -zugriff und sämtliche Kommunikationsprozesse herkömmliche Protokolle verwenden bzw. auf diesen basieren.
- Architekturen, mit denen Dienste definiert und bereitgestellt werden können.
- Verfahren, die auf so genannten mobilen Agenten basieren.

4.6.3.1 Architekturen auf Basis herkömmlicher Protokolle

Die zu dieser Kategorie gehörenden Dienstarchitekturen nutzen in der Regel als Basis bereits etablierte Netzwerk- und Anwendungsprotokolle. Der Vorteil besteht darin, dass weniger Zeit bei der Entwicklung benötigt wird und auf die Erfahrung mit bestehenden Protokollen zurückgegriffen werden kann. Außerdem ist die Implementierung solcher Systeme in bestehende Netzwerke einfacher durchzuführen, da keine Kompatibilitätsschwierigkeiten bestehen. Das Spektrum der Einsatzmöglichkeiten in diesem Bereich ist sehr groß und wächst gerade durch die aktuelle Tendenz zu personalisierten und kontextsensitiven Diensten ständig weiter. Die Techniken für solche Dienste entwickelten sich aus den ersten Architekturen für LBS. Dazu gehört z. B. das an der Universität von Lancaster entwickelte und unter anderem in [Born03] vorgestellte Guide-System. Hierbei werden dem Nutzer mit Hilfe eines mobilen Endgerätes (z. B. *Personal Digital Assistent* (PDA)) Informationen zu seiner

aktuellen Umgebung geliefert. Implementiert ist auch eine Navigation, die dadurch das ganze System zum virtuellen Touristenführer macht.

Aktuell erlangen neben Touristeninformationssystemen auch kontextsensitive Systeme, die behinderte Menschen im Alltag unterstützen, eine immer bedeutendere Rolle. Dort haben neben der Position auch andere Kontextinformationen großen Einfluss. In [DHST03] werden z. B. Profile für ein Touristisches Assistenzsystem (TAS) beschrieben, die Informationen zum Nutzer (z. B. Art und Grad der Behinderung) und zum Endgerät (z. B. Art, Displaygröße, Batteriestatus) widerspiegeln. Die dort vorgehaltenen Daten werden durch aktuelle Kontextinformationen zum Nutzer (z. B. Position, Wetter) ergänzt. Damit kann das System den Nutzer entsprechend seiner Behinderung und seines Kontextes navigieren und ihm angepasste Informationen zukommen lassen. In [Vosw04] werden ein solches Nutzer- und Geräteprofil sowie deren Implementierung beschrieben. Deutlich wird bei dieser Lösung die Komplexität der Informationsverwaltung, -darstellung und auch -verknüpfung. Andererseits führt die Entwicklung der Rechentechnik zu immer leistungsfähigeren Verfahren. Deshalb werden schon heute sehr flexible Architekturen entwickelt, die nicht nur eine bestimmte Zielgruppe bedienen, sondern allgemein dort einsetzbar sind, wo kontextsensitive Dienste angeboten werden. Dazu gehört beispielsweise die in [Lewa07] beschriebene Architektur SFINKS.

Ein weiteres aktuelles Beispiel aus der Forschung ist der Sonderforschungsbereich 627 „Umgebungsmodelle für mobile kontextbezogene Systeme“ (Nexus) [SFB06] der Deutschen Forschungsgemeinschaft (DFG). Der Fokus des dort geförderten Forschungsprojektes liegt auf der Entwicklung einer Plattform zur Bereitstellung von Kontextinformationen und zur Modellierung des Kontextes, sodass verschiedene Anwendungen darauf zugreifen und diese Daten nutzen können [Nick07]. Schwerpunkt bei Nexus bilden die LBS. Erwähnt wird dabei im Teilprojekt „Kontextbezogene Kommunikation“ [SFB07] auch eine kontextabhängige Vermittlung. Hierbei kann jedoch nicht von einer klassischen Dienstanfrage ausgegangen werden, da für die Versorgung der Clients mit Daten ein *Push*-Dienst mit Hilfe eines Overlay-Netzes auf Anwendungsebene verwendet wird [GeDü07]. Nachrichten, die als so genannte *Contextcasts* versendet werden, sind damit nur für Empfängergruppen mit einem bestimmten Kontext erreichbar. Da alle Knoten ihre Attribute den Routerinstanzen mitteilen müssen und Contextcasts die Empfänger nicht beliebig genau eingrenzen können, erzeugt diese Methode allerdings eine hohe Verkehrslast (siehe auch [GeDü07]). Mittlerweile besteht Nexus aus 17 Teilprojekten, was den Umfang der Thematik verdeutlicht. Zusammenfassend kann festgestellt werden, dass auch die Nexus-Plattform keine an das Routing gekoppelte Anfrage nach kontextsensitiven Diensten für einzelne Clients unterstützt.

4.6.3.2 Architekturen zur Dienstbereitstellung

Die bekanntesten Vertreter dieser Kategorie sind die *Open Services Gateway Initiative* (OSGi) [OSGi07] und die *Common Object Request Broker Architecture* (CORBA) [Obje04]. Diese Architekturen stellen offene Plattformen dar, über die Dienste bereitgestellt werden können. OSGi benötigt dazu eine *Java Virtual Machine* (JVM) und bietet die Möglichkeit zur Laufzeit dynamisch Anwendungen einzuspielen, zu aktualisieren und zu entfernen. OSGi wird häufig in der Haus- und Gebäudeautomatisierung sowie im Automotivbereich eingesetzt. CORBA stellt dagegen eine objektorientierte Middleware dar. Damit sollte ein einheitliches Rahmenwerk für

die Softwareentwicklung in verteilten Systemen geschaffen werden [ALSS03]. Um die Interoberabilität herstellerübergreifend unterstützen zu können, wird dort auf die Implementierung einer bestimmten Programmiersprache verzichtet. So können Dienstangebote in unterschiedlichen Sprachen programmiert und für unterschiedliche Betriebssysteme angeboten werden. Beide Plattformen arbeiten unabhängig von der Netzwerkschicht und bieten darüber hinaus selbst keine Dienstsuche, die daher über Zusätze erbracht werden muss.

4.6.3.3 Mobile Agenten

Mobile Agenten, die zur Dienstsuche verwendet werden, definiert [Wang02] folgendermaßen:

Definition 4.5 (Mobiler Agent) *„A mobile Agent is an identifiable software entity that consists of program code and the associated internal global and execution states, and has the ability to perform the task delegated by the user or another entity autonomously, and, of its own choosing, to move from one system to another within the network during execution.“*

Es handelt sich also um identifizierbare Software, die selbständig Aufgaben ausführen und sich währenddessen über das Netzwerk von einem System zum anderen bewegen kann. Ein Beispiel ist die in [Wang02] vorgestellte auf Agenten basierende Plattform CHAPLET, die für mobile Netzwerkumgebungen entwickelt wurde und auch eine kontextsensitive Suche erlaubt. Die Architektur ist für drahtlose Infrastrukturnetze entwickelt worden, funktioniert aber auch für Ad-hoc-Netz, sofern dieses an ein Festnetz angebunden ist. Ein solches System arbeitet auf den oberen Protokollschichten. Abhängig von der Größe der Agenten eines solchen Systems und inwieweit tatsächlich der Code im Netz übertragen wird, ist hierdurch eine Mehrbelastung des Netzwerkverkehrs zu erwarten. Der Weg des Agenten ist im Netzwerk nicht an bestimmte Knotenpunkte gebunden, sodass der Netzprovider auch hier keine Möglichkeit hat, Einfluss auf den Datenverkehr zu nehmen. Schließlich ist dieses Verfahren auch sehr ressourcenintensiv, weil auf jedem beteiligten Netzknoten die entsprechende Architektur implementiert und die zugehörigen Funktionen realisiert sein müssen. Wie in Abschnit 2.2 dargelegt, ist dies gerade für Knoten in Ad-hoc-Netzen problematisch, weil deren Energievorrat begrenzt ist.

Insgesamt können über die in diesem Abschnitt vorgestellten Architekturen folgende Schlüsse gezogen werden. Die Verfahren aus allen drei vorgestellten Kategorien arbeiten oberhalb der Netzwerkschichten. Sofern die Architekturen eine Dienstsuche unterstützen, basieren diese auf herkömmlichen Protokollen und damit gelten die Schlussfolgerungen aus Abschnitt 4.6.2. Sofern zur Dienstsuche zusätzliche Funktionen in die Architekturen und Dienstplattformen implementiert werden müssen, erhöht sich deren Komplexität. Dabei ist zu berücksichtigen, dass auf mobile Agenten basierende Architekturen sehr ressourcenintensiv sein können.

4.7 Kapitelzusammenfassung

Die Diskussion zur kontextsensitiven Diensterbringung hat gezeigt, dass es sich um ein sehr vielschichtiges Thema handelt. Zu Beginn des Kapitels ist deshalb auf den

Begriff Kontext eingegangen und dabei festgestellt worden, dass für die vorliegende Arbeit nur der Kontext des Nutzers von Interesse ist. Zur Beschreibung dieses Kontextes dienen die Kontexttypen und Kontextinformationen. Es konnte gezeigt werden, dass ein Kontext niemals vollständig beschrieben werden kann. Dies führt aber zu keinen Einschränkungen, weil lediglich so viele Informationen benötigt werden, dass ein Dienst optimal an den Nutzer angepasst werden kann. Des Weiteren wurde erläutert, wie der Kontext schließlich erfasst und verarbeitet werden muss, damit dessen Beschreibung möglich ist und die Kontextinformationen zur Erbringung eines Dienstes genutzt werden können. Der Begriff Dienst wurde diesbezüglich erläutert und für diese Arbeit um das Attribut „kontextsensitiv“ erweitert.

Einen wesentlichen Aspekt stellt die Dienstsuche dar, weil sie die Basis der eigentlichen Dienstnutzung ist. Es wurden deshalb verschiedene existierende Architekturen und Protokolle zur Dienstsuche untersucht, gegenübergestellt und verglichen. Im Ergebnis zeigte sich, dass die auf Netzwerkebene arbeitenden Verfahren, nur eine beschränkte Reichweite besitzen. Sie sind deshalb nicht für eine Dienstsuche über Netzgrenzen hinaus geeignet. Bei Verfahren, die auf Anwendungsebene arbeiten, ist dies von der jeweiligen Spezifikation abhängig. Von allen Verfahren wird der Kontext des Nutzers nicht oder kaum berücksichtigt. Des Weiteren hat der Netzwerkprovider keine Möglichkeit steuernd auf den Suchprozess und die anschließende Dienstkommunikation einzugreifen. Die Untersuchung weiterer Architekturen, die auf herkömmlichen Protokollen basieren, verschiedener Dienstplattformen und der Dienstbereitstellung mittels Agenten zeigte, dass auch diese Techniken keine befriedigende Lösung bieten.

Andererseits könnte aber ein kontextsensitives Routingverfahren die bereits vorhandenen Dienstarchitekturen und Dienstplattformen unterstützen und diesen als Netzwerk mit integrierter kontextsensitiver Dienstsuche dienen. Ein weiterer Vorteil der kontextsensitiven Dienstsuche auf Netzwerkebene besteht dabei in der Trennung von Dienstsuche und Diensterbringung. Damit wird die Flexibilität bezüglich des genutzten Netzwerkes und der unterstützten Dienstarten erhöht. Außerdem resultiert aus dieser Flexibiltät die Möglichkeit, ein Verfahren zur Dienstsuche zu entwickeln, das sowohl in Infrastruktur- als auch in Ad-hoc-Netzen arbeitet und eine netzübergreifende Dienstsuche unterstützt.

Offen bleibt auch die Frage, wie mit Hilfe eines solchen Verfahrens realisiert werden kann, dass Netzwerkprovider steuernd auf die Kommunikationsprozesse einwirken können und somit eine Skalierung des Netzwerkes möglich wird. Wie die Funktionsweise eines solchen Routingverfahrens aussehen könnte, wird im folgenden Kapitel diskutiert.

5. Verbindung von Dienstsuche und Routing

Eine wesentliche These in Kapitel 4 war, dass die Dienstsuche auf Routingebene effizienter als in der Anwendungsschicht gestaltet werden kann. Deshalb soll dieser Ansatz auch für die Entwicklung einer Architektur zur Suche nach kontextsensitiven Diensten verwendet werden. Das folgende Kapitel beschreibt, welche Einflüsse bei der Entwicklung eines solchen Verfahrens eine Rolle gespielt haben, welche bestehenden Ansätze untersucht wurden und wodurch diese sich unterscheiden. Die daraus resultierenden Ergebnisse führen zu einem neuen Ansatz, welcher das kontextsensitive Routing einbindet. Die Funktionsweise wird erläutert und mit Hilfe einer vergleichenden Gegenüberstellung bewertet.

5.1 Einführung

Derzeit existiert schon eine Vielzahl von verschiedenen (spezialisierten) Routingprotokollen, die für bestimmte Umgebungsbedingungen (z. B. Netzwerktechniken) entwickelt wurden. Um aber auf die Bedürfnisse des Nutzers netzübergreifend eingehen zu können, muss eine Lösung gefunden werden, die auch in heterogenen Umgebungen arbeitet. Sicherlich kann auch der schon in Kapitel 3.5 dargestellte komponentenbasierte Ansatz aus [LZHJ+06] zu einem optimierten Routing führen. Allerdings wird hierbei nicht auf den Kontext des Teilnehmers eingegangen. Vielmehr wird mit dem dort verwendeten CBR-Protokoll versucht, ein optimales Routingverfahren bezüglich der Eigenschaften der genutzten Netzwerktechnik anzubieten. Ziel dieser Arbeit soll jedoch ein allgemeingültiges kontextsensitives Routingverfahren sein, welches auch den Kontext des Teilnehmers unabhängig von der Netzwerktechnik berücksichtigt.

Ausgangspunkt der Untersuchung von Möglichkeiten für eine kontextsensitive Dienstsuche war die Fragestellung, wie diese in einem Ad-hoc-Netz gestaltet werden kann. Es konnte festgestellt werden, dass es dort nicht sinnvoll ist, eine Suche nach mobilen Diensten über die IP-Adresse des zugehörigen Servers durchzuführen. In der Regel steht die IP-Adresse bis zur Initialisierung eines Knotens im Ad-hoc-Netz noch nicht fest. Praktischer ist also eine direkte Suche nach einem Dienst. Dies

kommt schließlich auch den Nutzern entgegen, die nicht an der IP-Adresse eines Servers, sondern an den dort angebotenen Diensten interessiert sind. Bei näherer Betrachtung musste jedoch festgestellt werden, dass eine auf ein Ad-hoc-Netz beschränkte Sichtweise nicht ausreicht. So ist beispielsweise davon auszugehen, dass auch zukünftig viele Dienste, auf die mobile Nutzer zugreifen möchten, in einem Infrastrukturnetz angeboten werden. Im Zuge der derzeitigen Netzintegration ist es deshalb notwendig, ein solches Verfahren für heterogene Netzwerkumgebungen zu entwickeln. Netzintegration bedeutet hierbei die Zusammenführung verschiedener Netzwerktechniken. Aktuell erfolgt diese Vereinigung mit der Tendenz zu einem paketvermittelten Netz hin. Als gemeinsames Vermittlungsprotokoll dient hierbei IP. Damit wird den Nutzern eine einheitliche Schnittstelle für die Übertragung ihrer Daten zur Verfügung gestellt. Dennoch müssen die Eigenschaften der verschiedenen Netzzugangstechiken gerade im Bereich der Ad-hoc-Netze (siehe Kapitel 2) berücksichtigt werden. Durch deren dynamische Topologie stellen sie die größte Herausforderung bei der Entwicklung eines geeigneten Verfahrens zur Dienstsuche dar. Wie auch in Abschnitt 4.6 gezeigt, existieren bereits Verfahren für Ad-hoc-Netze, die auf Netzwerkebene arbeiten. Diese unterstützen jedoch keinen Dienstzugriff über eine administrative Domäne bzw. Netzgrenze hinweg.

Damit besteht also die Frage, wie ein Verfahren zur Dienstsuche theoretisch gestaltet werden muss, damit es die folgenden Anforderungen im Gegensatz zu den Verfahren aus Abschnitt 4.6.2 vollständig erfüllen kann:

- Unterstützung kontextsensitiver Dienste
- Dienstanfrage auf Netzwerkebene
- Unterstützung heterogener Netzwerkumgebungen
- Adressierung über den Dienst

5.2 Ansätze für eine kontextsensitive Dienstsuche

Der Entwicklung eines Verfahrens zur kontextsensitiven Dienstsuche, das die in Abschnitt 5.1 genannten Anforderungen erfüllen kann, gingen Untersuchungen zu zwei vielversprechenden Ansätzen voraus. Im Folgenden werden diese Ansätze vorgestellt. Es wird deren Funktionsweise erläutert und darauf eingegangen, welche Vor- und Nachteile sich daraus ergeben. Im Ergebnis zeigt sich, dass keines der beiden Verfahren die gestellten Anforderungen zufriedenstellend erfüllen kann. In Kombination bilden die Funktionen beider Ansätze jedoch die Basis der für diese Arbeit entwickelten Lösung zur Realisierung des kontextsensitiven Routings, welche dann in Abschnitt 5.3 vorgestellt wird.

5.2.1 Anycast

Anycast [PaMM93] stellt einen Adressierungstyp in Netzwerken dar. Es arbeitet auf den netzwerkbezogenen Protokollschichten und wird aktuell auch schon zum Abruf von Diensten verwendet. Bekanntestes Beispiel ist die Adressierung von Namensservern im DNS nach [Hard02]. Anycast bietet somit die Möglichkeit, Dienstanfragen in der Netzwerkschicht durchzuführen.

5.2.1.1 Funktionsweise

Beim Anycast wird eine Anfrage in das Netzwerk gesendet. Der erste Server, der die Anfrage erhält, beantwortet diese. Die Realisierung von Anycast kann sehr unterschiedlich sein. [PaMM93] beschreibt dazu verschiedene Möglichkeiten der Adressierung. Hierbei wird vorgeschlagen, Well-Known-Adressen aus Adressbereichen der Subnetze oder aus einer separaten Adressklasse (ähnlich dem Multicast) zu verwenden. Beide Ansätze können auch miteinander verbunden werden. Zum Beispiel besteht die Möglichkeit, 256 Adressen aus einem Klasse-C-Netz als Anycast-Adressen zu reservieren. Einem Server können daraus eine oder mehrere Anycast-Adressen zugeordnet werden. Für den in dieser Arbeit verfolgten Ansatz einer Dienstsuche müsste dann eine Anycast-Adresse mit genau einem Dienst korrelieren und die Dienstsuche über diese Adresse erfolgen. Die anschließende Kommunikation kann dagegen auch via Unicast erfolgen (siehe [WSWZ04]).

Je nach Realisierung sind beim Anycast erhöhte Anforderungen an das Routing zu stellen. Um Dienste erreichen zu können, die sich außerhalb einer administrativen Domäne befinden, müssen die dafür nötigen Routen z. B. durch entsprechende Routingprotokolle verbreitet werden. Beim *Global IP-Anycast* (GIA) [KaWr00] erfolgt dies auf Basis des BGP [ReLi95]. Innerhalb einer administrativen Domäne sind dagegen Vorkehrungen zu treffen, um Adresskonflikte durch ein Angebot gleicher Dienste zu vermeiden. In [PaMM93] wird beispielsweise vorgeschlagen, dazu das für die Adressauflösung verantwortliche *Address Resolution Protocol* (ARP) zu verwenden.

Obwohl der Nutzer beim Senden einer Dienstanfrage eine IP-Adresse verwendet, besitzt er kein Wissen darüber, von welchem Server der Dienst letztlich erbracht wird. Um kontextsensitive Dienste zu unterstützen, müsste ein Anycast mindestens um Informationen zu den vom Client bereitgestellten Kontexttypen erweitert werden. So könnte nach einem Server gesucht werden, der einen optimal auf den Nutzer und dessen Kontext angepassten Dienst anbietet.

5.2.1.2 Diskussion

Mit den genannten Voraussetzungen stehen technische Möglichkeiten zur Verfügung, um die in Abschnitt 5.1 geforderten Eigenschaften zu realisieren. Selbst der Forderung nach einer Dienstadressierung wird in gewisser Weise entsprochen. Wird davon ausgegangen, dass beim Anycast eine bestimmte IP-Adresse mit einem bestimmten Dienst korreliert, kann dies durchaus auch als Dienstadressierung bezeichnet werden. Es ist lediglich festzulegen, wie die Adressen einander zugeordnet sind. Das führt aber dazu, dass mindestens so viele Adressen vorgehalten werden müssen, wie auch Dienste existieren. Da die Anzahl der freien und fest zuzuordnenden Adressen gerade beim IPv4 sehr begrenzt ist, kann dadurch auch nur ein sehr beschränktes Angebot von Diensten vorgehalten werden. Sind nun bei der Auswahl eines passenden Servers auch die bei einer Anfrage zur Verfügung stehenden Kontexttypen zu berücksichtigen, bieten sich zwei Varianten an:

1. Die Kontextinformationen werden als Optionen innerhalb eines Anycasts übertragen.
2. Jeder Dienst bildet zusammen mit einer bestimmten Kombination aus Kontexttypen einen eigenständigen neuen Dienst, nach welchem dann die Anfrage erfolgt.

Bei der ersten Variante erfolgt die Suche nach einem Dienst, wie in Abschnitt 5.2.1.1 bereits beschrieben. Allerdings werden bei dieser noch keine Kontexttypen berücksichtigt. D. h., der auf das Anycast antwortende Server reagiert allein auf die Dienstinformation. Um die Kontexttypen auszuwerten, müssen deshalb zusätzliche Funktionen integriert werden. Solche Funktionen können z. B. durch die Server selbst übernommen werden. Dies entspricht allerdings nicht mehr dem ursprünglichen Anycast-Ansatz. Eine Suche nach kontextsensitiven Diensten auf Netzwerkebene ist damit nicht möglich.

Die zweite Variante verfolgt dagegen stringent den Anycast-Ansatz und entspricht der Forderung einer Dienstsuche auf Netzwerkebene. Hierbei ist ein Dienst incl. dessen unterstützte Kontexttypen an eine Well-Known-Adresse gebunden. Das bedeutet auch, dass gleiche Dienste, die unterschiedliche Kontexttypen unterstützen, verschiedenen Adressen zugeordnet sind. Durch die begrenzte Anzahl von IP-Adressen führt dies jedoch zu einer weiteren Verringerung möglicher adressierbarer Dienste. Für die folgenden Vergleiche mit den anderen in diesem Kapitel vorgestellten Ansätzen dient diese Variante der kontextsensitiven Dienstsuche via Anycast als Referenz.

Mehr Möglichkeiten für die Nutzung von Anycasts bietet das Vermittlungsprotokoll IPv6 aufgrund der neuen Adressstruktur. Dabei ist nicht festgelegt, welcher Algorithmus zur Realisierung benutzt wird. [WeCh04] gibt hier einen Überblick über verschiedene Ansätze. Trotzdem stellt sich auch hier die statische Bindung zwischen IP-Adresse und Dienst als Nachteil für den Nutzer bzw. Client dar. Diese sind auf das Wissen über die Well-Known-Adressen angewiesen. Dazu können zwar Listen vorgehalten werden, deren Pflege ist allerdings bedingt durch die Evolution der Dienste sehr aufwändig.

Aktuell existieren auch Lösungsansätze, die Anycast innerhalb eines Ad-hoc-Netzes ermöglichen. [WSWZ04] nutzt dazu für die betreffenden Geräte zwei IP-Adressen. Der dienstanbietende Knoten kann über eine zugeordnete Anycast-Adresse gesucht bzw. gefunden werden. Die Kommunikation mit dem Client erfolgt dann jedoch über seine eindeutige Unicast-Adresse. Durch zusätzliche Routeneinträge in einem Knoten kann ermittelt werden, welches Mitglied einer Anycast-Gruppe sich am nächsten zu ihm befindet. Zur prototypischen Umsetzung wurde in [WSWZ04] das Routingprotokoll AODV erweitert. Trotz dieser Umsetzung erfolgt auch hier eine statische Zuordnung von IP-Adresse zu Dienst bzw. kontextsensitivem Dienst. Hinzu kommt auch, dass dieses Verfahren lediglich in Ad-hoc-Netzen arbeitet. Infrastrukturnetze werden nicht unterstützt.

Ein Vorteil des Anycast-Ansatzes bestünde darin, dass durch den Einsatz mehrerer Server, die die gleichen Dienste anbieten, eine Lastverteilung realisiert werden kann. Des Weiteren stehen bei Ausfall eines Servers Alternativen zur Verfügung, was wiederum zu einer Erhöhung der Verfügbarkeit und Stabilität beiträgt.

5.2.2 Zentrale Datenbank

Eine Dienstsuche mit Hilfe von Anycasts ist, wie bereits begründet wurde, nicht sehr flexibel. Deshalb wurde im Rahmen dieser Arbeit nach einer anderen Lösung für die Dienstsuche geforscht. Im Folgenden wird gezeigt, dass die Verwendung einer zentralen Datenbank bessere Möglichkeiten für die Nutzung kontextsensitiver Dienste bieten kann.

5.2.2.1 Funktionsweise

Bei diesem Ansatz wird in einem Netzwerk eine Datenbank zur Verfügung gestellt, die sämtliche Dienstanfragen beantworten kann. Die Dienste müssen dort registriert werden. Darüber hinaus enthält die Datenbank Informationen zu den einzelnen Kontexttypen, die von den jeweiligen Diensten unterstützt werden. Möchte ein Client einen kontextsensitiven Dienst abrufen, sendet er seine Anfrage an den Datenbankserver. Dort wird anhand der mitgelieferten Kontexttypen der optimal auf den Nutzer angepasste Dienst ausgewählt. Daraufhin erhält der Client als Antwort die IP-Adresse des entsprechenden Servers. Alternativ kann auch der Datenbankserver selbst verschiedene Dienste anbieten.

Um den Datenbankserver erreichen zu können, ist nur eine IP-Adresse nötig. Da die bei der Dienstsuche verwendete IP-Adresse und der jeweilige Dienst nicht direkt aufeinander abgebildet sondern vollkommen unabhängig voneinander sind, kann ein sehr breites Spektrum von Diensten verwaltet werden. Gleiche Dienste, die aber unterschiedliche Kontexttypen unterstützen, sind differenzierbar. Des Weiteren können Algorithmen implementiert sein, sodass bestimmte Server bevorzugt ausgewählt werden.

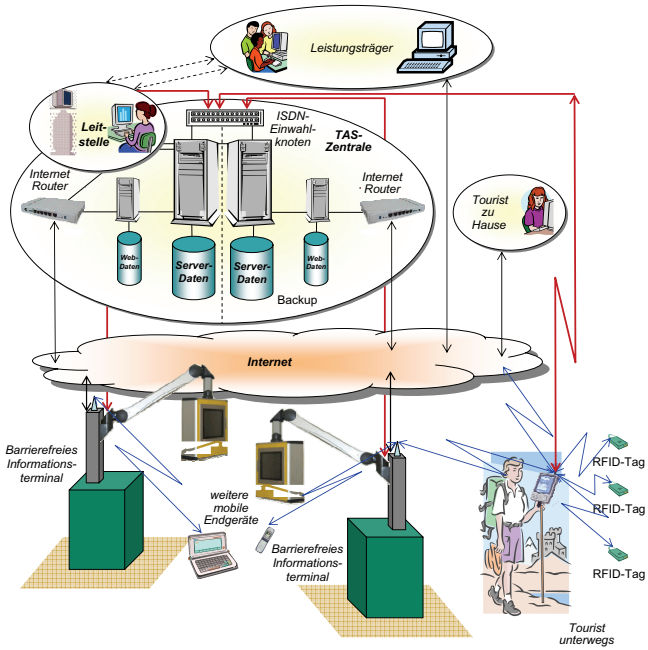


Abbildung 5.1: Kommunikationsarchitektur für das TAS nach [LeDS05]

Die zentrale Datenbank kann verschiedene Netzwerke bedienen. Abbildung 5.1 zeigt hierfür ein Beispiel. Dort ist die Architektur für das bereits erwähnte Projekt TAS dargestellt. Diese Datenbank kann über verschiedene Netzwerke abgerufen werden.

Dienstanbieter können sowohl über einen ISDN[1]-Router als auch über das Internet auf die Datenbank zugreifen, um ihre Angebote zu veröffentlichen. Die Dienstanfragen der Nutzer können darüber hinaus auch mobil über WLAN-Zugriffspunkte erfolgen. Wie in Abschnitt 4.6.2 gezeigt wurde, erfolgt eine Dienstanfrage gerade in einem Ad-hoc-Netz am effizientesten auf der Routingebene. Anstatt erst eine Route zu suchen und anschließend eine Dienstanfrage durchzuführen, werden beide Suchanfragen gleichzeitig gesendet. Deshalb ist die in Abbildung 5.1 dargestellte Kommunikationsarchitektur noch zu modifizieren, um die Anforderungen – Dienstanfrage auf Netzwerkebene – aus Abschnitt 5.1 zu erfüllen und damit auch Ad-hoc-Netze optimal zu unterstützen. Zur Dienstsuche beim aktuellen Ansatz ist das Auffinden der Datenbank nötig. Die anschließende Kommunikation erfolgt dann über höhere Schichten. Ansonsten erfüllt dieser Ansatz ebenfalls wie das Anycast alle übrigen Anforderungen. Es können damit auch heterogene Netzwerke bedient werden. Kontextsensitive Dienste werden unterstützt. Daneben erfolgt auch die Suche nach einem passenden Dienstanbieter über den Dienst selbst. Ein passender Dienstanbieter wird nur mit Hilfe der Informationen zum gewünschten Dienst und den verfügbaren Kontexttypen in der Datenbank gesucht und ausgewählt.

5.2.2.2 Diskussion

Der Ansatz „zentrale Datenbank“ ist zwar sehr flexibel bezüglich der Diensterbringung, hat aber den entscheidenden Nachteil, dass das System insbesondere in Ad-hoc-Netzen nicht sehr zuverlässig arbeitet. Schon der Verlust der Route zwischen Netz und System oder ein kleiner Fehler in der Datenbank kann dazu führen, dass es zu einem netzweiten Systemausfall kommt. Dem kann durch Spiegelung der Datenbank (siehe Abbildung 5.1) entgegengewirkt werden. Es ist dabei jedoch zu berücksichtigen, dass die gespiegelten Systeme entweder mit der gleichen Adresse arbeiten oder die jeweiligen Adressen der gespiegelten Systeme den Clients manuell übergeben bzw. in deren Konfiguration berücksichtigt werden müssen. Allerdings ist dann der administrative Aufwand und die Gefahr von Inkonsistenzen zwischen den Datenbanken entsprechend höher. Die Kosten für Anschaffung und Betrieb steigen entsprechend der Zahl der Spiegelungen. Existiert tatsächlich nur eine Datenbank, kommt hinzu, dass sich die ganze Last für die Bearbeitung der Dienstanfragen auf die zentrale Datenbank konzentriert. Ebenso kommt es zu einer Verkehrskonzentration im Umfeld der Datenbank, was durchaus zur Bildung von Flaschenhälsen führen kann. Diese Eigenschaften machen das System damit auch anfällig für Angriffe.

5.3 Ein neuer Ansatz – Router mit Kontextwissen

Beide zuvor vorgestellten Ansätze zur Dienstsuche erfüllen grundsätzlich die in Abschnitt 5.1 beschriebenen Anforderungen. Trotzdem ergeben sich aus der Funktionsweise auch Nachteile bezüglich der Flexibilität und Verfügbarkeit. Während sich beim Anycast die Abhängigkeit zwischen IP-Adresse und Dienst als unflexibel darstellt, kann in einem Ad-hoc-Netz die Verfügbarkeit einer zentralen Datenbank nicht ausreichend gewährleistet werden. Deshalb wurde ein Ansatz entwickelt, in dem die Nachteile überwunden und dafür die Vorteile beider Verfahren zusammengeführt sind.

[1]ISDN *Integrated Services Digital Network*

Wie bereits erläutert, ist eine Dienstsuche dann am flexibelsten, wenn diese in Verbindung mit den Vermittlungsfunktionen realisiert wird. Des Weiteren wurde festgestellt, dass gerade im Ad-hoc-Netz eine Dienstsuche dann am effizientesten ist, wenn diese in Kombination mit dem dort eingesetzten Routingverfahren erfolgt. Wird dann bei der Festlegung der Route zur Kommunikation zwischen Client und Server auch der Kontext des Teilnehmers berücksichtigt, führt dies zum kontextsensitiven Routing.

5.3.1 Kontextsensitives Routing

Das kontextsensitive Routing setzt sich aus zwei Bestandteilen zusammen – einer Dienstsuche und einer Wegewahl. Basierend auf der Suche nach einem kontextsensitiven Dienst erfolgt die Vermittlung eines Servers, der diesen Dienst anbietet. In Abschnitt 4.5 wurde gezeigt, dass der Dienst die zu seiner Erbringung obligatorischen und optionalen Kontexttypen bestimmt. Da die zugehörigen Kontextinformationen dynamisch sein können, müssen sie ggf., wie in Abschnitt 4.4 diskutiert, in bestimmten Abständen aktualisiert werden. Das Routing wird hierdurch nicht beeinflusst, weil der ursprünglich gewählte Server auch bei Änderung der Kontextinformation des Nutzers beibehalten wird. Daraus ist zu schlussfolgern, dass zur Suche eines kontextsensitiven Dienstes lediglich die zu seiner Erbringung genutzten Kontexttypen berücksichtigt werden müssen. Damit ergibt sich folgende Definition:

Definition 5.1 (Kontextsensitives Routing) *Kontextsensitives Routing ist eine Form der Wegewahl in einem paketvermittelten Netzwerk, bei der die Entscheidung für einen Weg (Route) zu einem dienstanbietenden Netzknoten (Server) in Abhängigkeit von dem gesuchten Dienst und den zu seiner Erbringung vom dienstsuchenden Netzknoten (Client) bereitgestellten Kontexttypen erfolgt.*

Um eine allgemeingültige Definition angeben zu können, wurden die Kontexttypen und damit der Kontext noch nicht näher spezifiziert. Mit Rücksicht auf Abschnitt 4.2 sei aber noch einmal darauf verwiesen, dass für die vorliegende Arbeit primär der Nutzerkontext von Interesse ist.

5.3.2 Funktionsweise

Im Folgenden wird erläutert, welche Funktionsweise sich hinter dem kontextsensitiven Routing verbirgt, welche Voraussetzungen dazu ein Netzwerk erfüllen muss und welche Unterschiede sich gegenüber den anderen Ansätzen ergeben.

5.3.2.1 Allgemein

In Abbildung 5.2 ist die Funktionsweise des kontextsensitiven Routing anhand eines einfachen Beispiels dargestellt. Dort befinden sich zwei Nutzer in einem Ad-hoc-Netz. Die Dienstanbieter (Server) sind dagegen über das Internet und das herkömmliche Fernsprechnetz/Intelligente Netz (IN) erreichbar. Die zwei Nutzer wollen nun den gleichen Dienst abrufen (z. B. Bestellen eines Taxis). Sie unterscheiden sich bezüglich ihres Kontextes nur in der Art der Behinderung. Andere Kontexttypen wie beispielsweise die Position sollen vorerst aus Gründen der Einfachheit nicht mit berücksichtigt werden. Nutzer 2 ist im Gegensatz zu Nutzer 1 auf einen Rollstuhl angewiesen.

Beide senden ihre Anfrage nach dem Dienst in das Netzwerk und werden an einen Knoten weitergeleitet, der diese Anfragen unter Berücksichtigung der mitgelieferten Kontexttypen bearbeiten kann. Für Nutzer 1 wird kein spezielles Angebot benötigt. Deshalb wird eine Route ausgewählt, die seine Anfrage an den nächsten in Frage kommenden Server (z. B. allgemeine Taxiangebote) weiterleitet. Dagegen wird für den Nutzers 2 eine Route zu einem Server ausgewählt, die zwar über eine höhere Anzahl von Hops erreichbar ist, aber dafür zu einem Dienst führt, der optimal auf den Nutzer angepasst ist (z. B. Taxidienste für behindertengerechte Transporte).

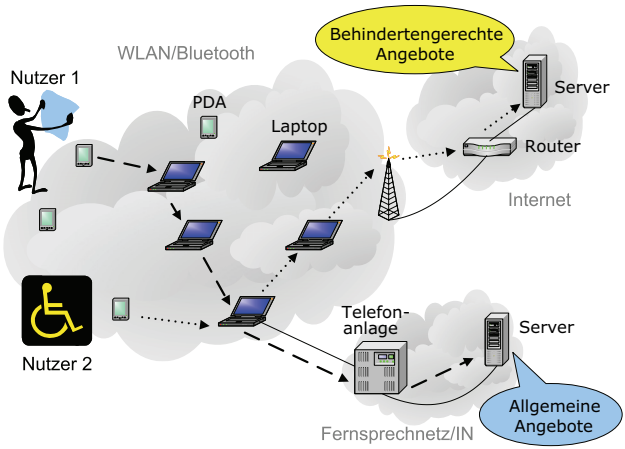


Abbildung 5.2: Beispielszenario für das kontextsensitive Routing

Wie aus dem vorgestellten Szenario leicht zu erkennen ist, müssen in einem Netzwerk spezielle Router implementiert sein, die die Dienstanfragen bearbeiten können. Auf deren Funktionsweise soll nun eingegangen werden.

5.3.2.2 Kontextrouter

Knoten, die das kontextsensitive Routing unterstützen, werden im Folgenden als Kontextrouter bezeichnet. Einen ersten Ansatz für die Funktionsweise eines solchen Routers wurde bereits in [DeSe03] vorgestellt. Abbildung 5.3 zeigt eine präzisierte Darstellung. Sie umfasst die wesentlichen Bestandteile, die für eine kontextsensitive Dienstsuche realisiert werden müssen. An dem Router bzw. Kontextrouter ist eine unbestimmte Anzahl von Clients angeschlossen, mit Hilfe derer die Nutzer auf die kontextsensitiven Dienste zugreifen können. Zur Diensterbringung steht ebenfalls eine unbestimmte Anzahl an Servern bereit.

Damit der Kontextrouter Anfragen von Nutzern korrekt bearbeiten kann, benötigt er Informationen über die im Netzwerk vorhandenen Dienste. Dazu müssen die entsprechenden Server mit den dort angebotenen Diensten und den zugehörigen Kontexttypen auf dem Router registriert werden. In Abbildung 5.3 wird dies als Kontextwissen beschrieben. Da die Dienstsuche auf der Routingschicht erfolgen soll, müssen die dort verwendeten Pakete um mindestens zwei Informationen erweitert werden. Wie in Abbildung 5.4 dargestellt, werden zusätzlich zu den allgemeinen Adressinformationen

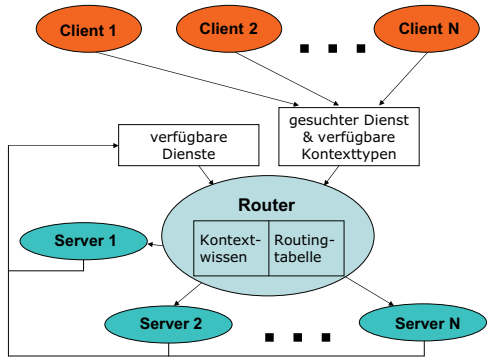


Abbildung 5.3: Funktionsweise des Kontextrouters

der gesuchte Dienst und die zu den verfügbaren Kontextinformationen gehörenden Kontexttypen übertragen. Die allgemeine Adressinformation wird benötigt, um den Kontextrouter erreichen zu können. Sofern das als Basis dienende Routingprotokoll weitere Optionen unterstützt oder ggf. Optionen für das Kontextrouting vorgesehen werden, sind diese in der Paketstruktur ebenfalls zu berücksichtigen.

Adresse (Kontextrouter)	Dienst	Kontexttypen	ggf. Optionen

Abbildung 5.4: Aufbau des modifizierten Routingpaketes

Der Kontextrouter kann jetzt die Dienstanfragen der Clients mit Hilfe seines Kontextwissens auswerten und den Server auswählen, der den geeignetsten Dienst für den Nutzer anbietet. Dabei werden nur die registrierten Dienste berücksichtigt, die mit dem angefragten überstimmen. Darüber hinaus muss jedoch auch ein Vergleich zwischen den vom jeweiligen Dienst unterstützten und den vom Client gelieferten Kontexttypen erfolgen. Konnte durch den Router ein passender Server gefunden werden, erhält der Client die entsprechenden Informationen darüber. Er ist nun in der Lage auf den kontextsensitiven Dienst zuzugreifen.

Damit die Server und Clients einen Kontextrouter finden können, muss eine geeignete Methode zur Verbreitung seiner Adresse verwendet werden. Hierbei ist zu beachten, dass die Vorteile des kontextsensitiven Routings erhalten bleiben. Es ist dem Nutzer beispielsweise nicht zuzumuten, dass er die Adressen der ihm zur Verfügung stehenden Router kennen muss. Da die Verbreitung der Adresse jedoch unabhängig vom kontextsensitiven Routing selbst erfolgt, soll an dieser Stelle nicht näher darauf eingegangen werden. Bei der Entwicklung einer Architektur zum kontextsensitiven Routing muss dies aber unbedingt berücksichtigt werden. In Abschnitt 6.3.1 wird diese Thematik deshalb ausführlich behandelt.

5.3.3 Diskussion

In den beschriebenen Szenarien wurde bis jetzt lediglich ein einzelner Kontextrouter verwendet. Da es sich jedoch um einen erweiterten herkömmlichen Router handelt, kann dessen Anzahl und damit die Verfügbarkeit beliebig erhöht werden. Dadurch

wird dieses Verfahren besonders interessant für Ad-hoc-Netze, wo nach Definition jeder Netzknoten auch als Router arbeitet. Durch Verwendung mehrerer Router mit Kontextwissen könnte hier eine einfache Erhöhung der Robustheit bzw. Verfügbarkeit erfolgen. Außerdem sind Algorithmen vorstellbar, die eine Kommunikation zwischen den Routern erlauben und somit beispielsweise auch eine Lastverteilung innerhalb des Netzwerkes unterstützen. Während sich bei der in Abschnitt 5.2.2 beschriebenen zentralen Datenbank einerseits Probleme durch die Größe der Datenbank selbst und anderseits durch den eingeschränkten Adresspool ergeben können, wirken sich diese Eigenschaften beim kontextsensitiven Routing weniger negativ aus. Es müssen hier lediglich die Routinginformationen vorgehalten werden, wobei Inkonsistenzen deswegen keine Rolle spielen, weil alle Kontextrouter unabhängig und individuell betrieben werden. Jeder Server ist dabei selbst für die Registrierung seiner Dienste verantwortlich. Außerdem kann der Netzbetreiber die Adressen der Kontextrouter frei wählen, ohne dass diese den Nutzern bekannt sein müssen, da vorgesehen ist, die Adressinformationen automatisch im Netzwerk zu verbreiten.

Natürlich werden sämtliche Vorteile durch eine Erhöhung der Komplexität der Router erkauft. Allerdings wird die Prozessortechnik immer leistungsfähiger, sodass sich das kontextsensitive Routing nur unwesentlich auf die Performance der Hardware bzw. des Netzwerkes auswirken wird. Ergänzend soll auch erwähnt werden, dass eine Erhöhung der Komplexität der Routerfunktionen eines Knotens im Ad-hoc-Netz gleichbedeutend mit einem erhöhten Energieverbrauch ist. Das wiederum schränkt die Laufzeit des Knotens ein. Andererseits betreffen diese Probleme auch die beiden anderen Ansätze, sodass dies im direkten Vergleich keinen Nachteil darstellt.

Das kontextsensitive Routing baut auf einem Routingprotokoll auf. Unabhängig davon, ob bei einer Umsetzung ein neues Protokoll entwickelt oder ein bereits bestehendes verwendet wird, muss das Konzept auch die in Abschnitt 3.3 aufgeführten unbedingten Anforderungen bezüglich des Routings erfüllen.

5.4 Vergleich und Bewertung

Das kontextsensitive Routing und der damit verbundene Einsatz von Kontextroutern vereint die Vorteile aus Anycast-Ansatz und zentraler Datenbank ohne dabei deren Nachteile zu übernehmen. Tabelle 5.1 vergleicht dazu noch einmal alle drei Ansätze. Von den ursprünglichen Anforderungen an ein Verfahren zur Dienstsuche aus Abschnitt 5.1 wurden die Dienstanfrage auf Netzwerkebene, die Unterstützung heterogener Netzwerke und die Möglichkeit der Dienstadressierung bewertet. Da alle drei Ansätze gleichermaßen die Suche nach einem kontextsensitiven Dienst unterstützen, gilt diese Anforderung als gleichwertig erfüllt. Zusätzlich stellt die Tabelle weitere grundsätzliche Eigenschaften gegenüber, in denen sich die Routingansätze unterscheiden. Dabei wird jeweils der schlechteste Ansatz mit (–), der beste mit (+) der sich dazwischen befindende Ansatz mit (o) bewertet.

Die Tabelle ist folgendermaßen zu interpretieren:

Dienstanfrage: Lediglich beim Anycast und beim kontextsensitiven Routing erfolgt die Dienstanfrage auf Netzwerkebene. Für die zentrale Datenbank ist eine entsprechende Erweiterung nötig.

Ansatz	Dienst-anfrage	heterogene Netzwerke	Dienst-adressierung	Flexi-bilität	Robust-heit	Effi-zienz	Skalier-barkeit
Anycast	+	o	–	–	+	–	o
zentrale Datenbank	–	–	+	+	–	–	–
Kontext-router	+	+	+	+	+	+	+

Tabelle 5.1: Vergleich der Routingansätze

heterogene Netzwerke: Zwar werden von allen drei Ansätzen heterogene Netzwerke unterstützt, allerdings ist der dazu nötige Aufwand sehr unterschiedlich. Der Anycast-Ansatz bietet Lösungen für Infrastruktur- und Ad-hoc-Netze. Für einen gleichzeitigen Betrieb in beiden Netzwerken sind jedoch zusätzliche Maßnahmen nötig. Die zentrale Datenbank kann theoretisch in einem Ad-hoc-Netz verwendet werden. Praktisch ist der Zugriff auf diese jedoch durch den zentralen Ansatz sehr eingeschränkt. Die Kontextrouter können ohne Einschränkungen in heterogenen Netzwerken arbeiten.

Dienstadressierung: Hierbei wird unterschieden, inwieweit ein Dienst an eine IP-Adresse gebunden ist. Am flexibelsten gestalten sich hier die Ansätze der zentralen Datenbank und des kontextsensitiven Routings, weil die IP-Adressen der Server vollkommen unabhängig von den dort angebotenen Diensten sind. Beim Anycast-Ansatz ist der kontextsensitive Dienst direkt an eine Anycast-Adresse gebunden. Die Adressierung der Server ist damit von vornherein eingeschränkt. Vor allem in IPv4-Netzwerken kann damit das mögliche Dienstangebot auf Grund der beschränkten Anzahl von Anycast-Adressen im Verhältnis zu den anderen Ansätzen nur sehr klein sein.

Flexibilität: Diese Eigenschaft bewertet den Einfluss auf die Dienstsuche, wenn beispielsweise neue Dienste eingeführt werden und alte Dienste wegfallen. Hier besteht ein unmittelbarer Zusammenhang zur Adressierung. Systeme, deren Server unabhängig zum angebotenen Dienst adressiert werden, sind bei Änderung des Dienstangebotes viel flexibler als solche, die Well-Known-Adressen verwenden.

Robustheit: Die Robustheit bezieht sich auf die Ausfallsicherheit bzw. Fehleranfälligkeit der einzelnen Ansätze. Wie bereits beschrieben, bietet dafür ein zentraler Ansatz die wenigsten Möglichkeiten. Bei Anycast und kontextsensitivem Routing existieren dagegen keine Beschränkungen. Hier kann die Robustheit mit steigender Zahl von Servern bzw. Kontextroutern beliebig erhöht werden.

Effizienz: Die Effizienz bewertet aus Sicht des Routings, welcher Ansatz sich bei der Suche nach einem Dienst mit Rücksicht auf die Anforderungen eines Ad-hoc-Netzes als am effektivsten zeigt. Beim Anycast und auch bei der zentralen Datenbank ist die Suche nach einer Route zum Server bzw. zur Datenbank unabhängig von der Dienstsuche. D. h., es sind mindestens zwei Kommunikationsprozesse (Routensuche und Dienstanfrage) nötig. Im Gegensatz dazu wird die Dienstanfrage beim kontextsensitiven Routing schon bei der Suche nach einem Weg zum Kontextrouter mitgesendet.

Skalierbarkeit: Diese Eigenschaft bewertet, inwieweit es möglich ist, den jeweiligen Ansatz an die Bedürfnisse des Netzwerkes (Anzahl der Clients, Server, Verkehr usw.) anzupassen. Die zentrale Datenbank kann zwar eine Lastverteilung bei den registrierten Servern vornehmen, sie selbst stellt aber bei Dienstanfragen einen Flaschenhals dar. Mit Anycast kann eine Lastverteilung realisiert aber nur schwer gesteuert werden. Das kontextsensitive Routing ist aufgrund des Wissens über die verfügbaren Server im Netzwerk sehr gut skalierbar. Eine gezielte Steuerung ist dabei durchaus möglich. Zusätzlich kann der Netzwerkverkehr über die Zahl der Kontextrouter und der dort vorgehaltenen Daten über kontextsensitive Dienste und den zugehörigen Servern skaliert werden.

5.5 Kapitelzusammenfassung

Das vorliegende Kapitel hat zwei Ansätze für eine Dienstsuche, die die Hauptanforderungen nach „Unterstützung kontextsensitiver Dienste“, „Unterstützung heterogener Netzwerkumgebungen“ und „Adressierung über den Dienst“ erfüllen, gegenübergestellt. Im Gegensatz zum Anycast erfüllt dabei die zentrale Datenbank nicht die Anforderung nach einer „Dienstanfrage auf Netzwerkebene“. Trotzdem weist der Datenbankansatz gerade in Bezug auf Adressierung und Flexibilität Vorteile auf, die bei der Entwicklung einer geeigneteren Architektur mit zu berücksichtigen sind.

Die Funktionsweisen der beiden Ansätze wurden beschrieben. Ihre Vor- und Nachteile wurden aufgezählt und diskutiert. Die daraus resultierenden Ergebnisse verbunden mit den Erkenntnissen aus Kapitel 4 führten zu einer neuen Lösung. Diese basiert auf Routern mit Kontextwissen. Ein solcher Kontextrouter kann bei einer Suche nach einem kontextsensitiven Dienst einen auf die Bedürfnisse des Nutzers optimal angepassten Dienst bzw. Server finden. Die sich daraus ergebende Wegewahl wird als kontextsensitives Routing bezeichnet. Die notwendigen Funktionen wurde beschrieben und diskutiert. Mit dieser Lösung wird es möglich, alle Hauptanforderungen an ein Verfahren zur kontextsensitiven Dienstsuche zu erfüllen. Im anschließenden Vergleich aller drei Ansätze erwies sich das kontextsensitive Routing als am vielversprechendsten. Es vereint die Vorteile von Anycast und zentraler Datenbank, ohne deren diskutierten Nachteile zu übernehmen. Lediglich eine gewisse Erhöhung der Komplexität kann als negativ empfunden werden. Dafür überwiegen aber die Vorteile bezüglich der Flexibilität, Robustheit, Effizienz und Skalierbarkeit. Da der Ansatz auf Routingebene realisiert wird, ist er transparent bei der Einbindung in bestehende Systeme bzw. interoperabel zu neuen Systemen, sofern diese IP unterstützen.

Damit ist das kontextsensitive Routing eine vielversprechende Alternative zur herkömmlichen Dienstsuche. Prinzipiell kann dieser Ansatz auch die in Tabelle 4.2 aufgeführten Eigenschaften erfüllen. Das hängt jedoch von dem endgültigen Konzept für eine praktische Realisierung ab. Ein Vorschlag dazu wird im folgenden Kapitel diskutiert.

6. Architekturkonzept

Der in Abschnitt 5.3 vorgeschlagene Lösungsansatz des kontextsensitiven Routings wird derzeit von keinem Routingverfahren unterstützt. Auch die in Abschnitt 4.6 vorgestellten aktuellen Architekturen zur Dienstsuche basieren auf anderen Mechanismen. Dort wird weder eine Suche nach kontextsensitiven Diensten unterstützt, noch sind diese Verfahren für heterogene Netzwerke geeignet. Hinzu kommt, dass die meisten Verfahren auf der Anwendungsschicht und somit in Ad-hoc-Netzen ineffizient arbeiten. Mit der Entwicklung eines Konzeptes zur Realisierung des kontextsensitiven Routings soll eine flexible Architektur zur Dienstsuche geschaffen werden, die die beschriebenen Nachteile behebt und auch heterogene IP-Netzwerkumgebungen unterstützt.

Die folgenden Abschnitte beschreiben detailliert einen Vorschlag für eine solche Architektur. Dazu wird in Abschnitt 6.1 zunächst deren Aufbau und Funktionsweise eingeführt, die benötigten Netzwerkkomponenten vorgestellt und ihre Aufgaben erläutert. In Abschnitt 6.2 erfolgt eine vergleichende Gegenüberstellung verschiedener Routingprotokolle, um das für die Architektur geeignetste Protokoll auswählen zu können. Daraufhin werden die Funktionsweise dieses Protokolls und daran durchzuführende notwendige Erweiterungen für das kontextsensitive Routing erläutert. Abschnitt 6.3 beschreibt detailliert, welche Funktionen in den einzelnen zur Architektur gehörenden Komponenten realisiert werden müssen und wie die dafür notwendigen Modifikationen der entsprechenden Protokolle aussehen. Abschließend wird das entwickelte Konzept in Abschnitt 6.4 bewertet und mit den in Abschnitt 4.6 aufgeführten Verfahren verglichen.

6.1 Einführung

Das Gesamtkonzept der Routingarchitektur ist in Abbildung 6.1 dargestellt. Das Netzwerk wird symbolisch durch drei Subnetze repräsentiert, die über den Kontextrouter miteinander verbunden sind. Der Kontextrouter stellt dabei die zentrale Komponente für die Kommunikation dar. Das Konzept unterstützt aber auch die Möglichkeit, dass mehrere Kontextrouter gleichzeitig dieselben Subnetze bedienen oder sich innerhalb eines einzelnen Ad-hoc-Netzes befinden können. Über die in der

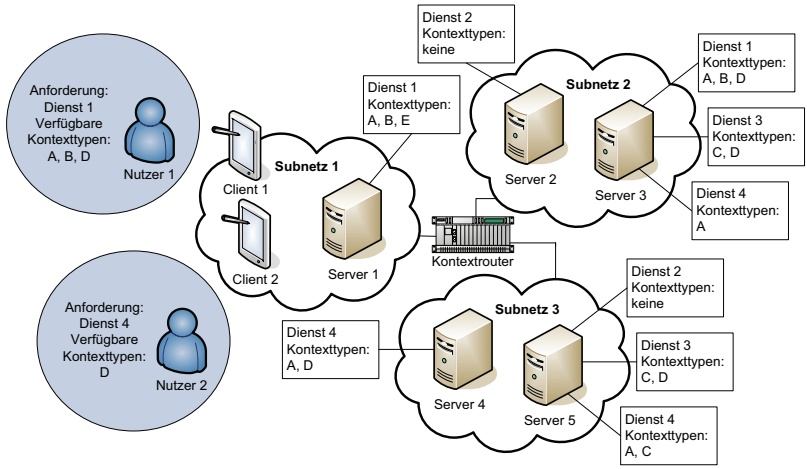


Abbildung 6.1: Die Komponenten der kontextsensitiven Routingarchitektur

Abbildung dargestellten Clients greifen die Nutzer auf die Dienste zu, welche von den Servern angeboten werden. Der Begriff Server ist an dieser Stelle zu erweitern. Im Allgemeinen wird damit ein System bezeichnet, das einen Dienst anbietet. Im Speziellen können bei den Servern jedoch drei Typen unterschieden werden. Dazu gehören Server, die:

- nur herkömmliche Dienste anbieten (Server 2 in Abbildung 6.1).
- die sowohl kontextsensitive als auch herkömmliche Dienste unterstützen (Server 5 in Abbildung 6.1).
- die ausschließlich kontextsensitive Dienste bereitstellen (Server 1, 3 und 4 in Abbildung 6.1).

Die beiden letzteren Typen werden auch als kontextsensitive Server bezeichnet. Die Unterscheidung der einzelnen Server ist genau dann wichtig, wenn auf eine Anfrage eines Clients hin der passende Dienst gefunden und zugeordnet werden muss. Um dies realisieren zu können, müssen sich die kontextsensitiven Server beim Kontextrouter registrieren und die jeweils unterstützten Kontexttypen angeben.

Möchte nun ein Nutzer einen kontextsensitiven Dienst verwenden, ruft er diesen mit seinem Gerät ab. Dazu wird auf dem Client eine entsprechende Anwendung gestartet, die eine Anfrage mit Angabe der von ihm unterstützten Kontexttypen an einen Kontextrouter sendet. Der Kontextrouter wertet anhand des angeforderten Dienstes und der mitgesendeten Kontexttypen aus, welcher Server als Diensterbringer in Frage kommen kann. Wurde ein passender Server gefunden (siehe Abbildung 6.1: Nutzer 1 $\Rightarrow$ Server 3, Nutzer 2 $\Rightarrow$ Server 4) und dieser durch den Client akzeptiert, kann die weitere Kommunikation nach zwei verschiedenen Methoden erfolgen.

1. Der Router kann die nun folgenden Dienstanfragen direkt an den Server weitersenden und im weiteren Verlauf als Proxy zwischen Client und Server arbeiten. Dieses Szenario kann auch als „Proxykommunikation“ betrachtet werden.

2. Der Router kann die Adresse des gefundenen Servers an den Client zurückgeben. Der Client kann dann direkt mit dem Server kommunizieren, deshalb wird dies als „direkte Kommunikation“ bezeichnet.

Der Nutzer bemerkt keinen Unterschied zwischen beiden Varianten. Die Vor- und Nachteile der beiden Methoden werden im Folgenden erläutert.

Bei der Proxykommunikation kann sofort auf Serverausfälle reagiert werden. Ist ein Server nicht mehr erreichbar, wählt der Router anhand der gespeicherten Daten einen anderen Server, der aber den gleichen Dienst anbieten muss. Diese Auswahl ist prinzipiell zu jedem Zeitpunkt möglich. Damit können auch auf einfache Weise Backuplösungen implementiert werden. Die Flexibilität des Systems lässt sich damit so weit erhöhen, dass in bestimmten Abständen überprüft werden kann, ob der verwendete Server noch optimal für die aktuell zu berücksichtigenden Kontexttypen ist oder ob ein neu ins Netz hinzugekommener Server bessere Ergebnisse liefert. In diesem Fall ist dann ein Wechsel zu dem anderen Server möglich. Damit erhält der Nutzer immer den für ihn am besten angepassten Dienst. Der Server bleibt für den Nutzer zu jedem Zeitpunkt „unsichtbar“. Da der gesamte Verkehr der Kommunikation über den Router geführt wird, kann dieser auch Verwaltungs- und Monitoringfunktionen übernehmen. An dieser Stelle wären z. B. Funktionen zum Abrechnungsmanagement denkbar. Die Proxykommunikation hat aber auch Nachteile. So erhöht sich beispielsweise die Komplexität des Routers, da dieser ein Großteil der Funktionalität für die Kommunikation erbringen muss. Das kann seine Rechenleistung beeinträchtigen. Da der Server „unsichtbar“ ist, hat der Nutzer nur begrenzten Einfluss auf dessen Auswahl. Dies könnte bei unseriösen Providern durchaus zu Problemen führen.

Fällt dagegen während einer direkten Kommunikation ein Server aus, muss erst erneut eine Anfrage an den kontextsensitiven Router gesendet werden. Dieser wählt einen neuen Server aus, der am besten für den Nutzer geeignet ist, und sendet dessen Adresse an den Client zurück. Nun kann dieser wieder direkt mit dem Server kommunizieren. Entsprechend umständlich würde sich dann auch der Serverwechsel gestalten, wenn sich während einer laufenden Kommunikation die Anforderungen bezüglich der Kontexttypen ändern. Der Nutzer hat dafür jedoch die Möglichkeit bei der Wahl des Servers ggf. einzuschreiten. Durch die gelieferte Zieladresse kann der Server identifiziert werden. Wie in den folgenden Abschnitten noch gezeigt wird, lässt sich dies aber auch mit der Proxykommunikation realisieren. Vorteilhaft ist die geringere Belastung der Router, was gerade bei Ad-hoc-Netzen sinnvoll ist, da die dortigen Netzknoten nur begrenzte Energiereserven besitzen. Jedoch verlagert sich jetzt die Intelligenz zur Realisierung des Algorithmus zum Client. Dort muss die Routingsoftware auch die zusätzlichen Kommunikationsabläufe übernehmen.

Im Gegensatz dazu ist anzumerken, dass zum einen die Anzahl der Kontextrouter abzählbar sein wird. Zum anderen werden diese mit großer Wahrscheinlichkeit vorzugsweise an den Netzübergängen eingesetzt werden, d. h., es handelt sich um durchaus leistungsfähige Netzknoten. Diese verbinden auch weiterhin Netze bzw. Subnetze miteinander. Damit kann beim Einsatz einer Proxykommunikation bei den Clients

Energie gespart werden, da der höhere Rechenaufwand bei den Routern liegt. Nachdem der Client einen Vorschlag vom Kontextrouter erhalten hat, steht dem Provider keine Möglichkeit mehr zur Verfügung, den Verkehr zu monitoren oder steuernd auf die Kommunikation einzuwirken

Wie aufgezeigt wurde, haben beide Kommunikationsvarianten ihre Vor- und Nachteile. Aufgrund der Möglichkeiten zum Monitoring und des Kostenmanagements wurde sich für die Methode „Proxykommunikation“ entschieden. Hinzu kommen die Robustheit bzw. Backupmöglichkeiten bei Ausfall eines Servers. Hier kann mit der Proxykommunikation schnell reagiert werden, ohne dass der Nutzer oder Client etwas von diesem Ausfall bemerkt.

Im folgenden Abschnitt wird untersucht, welches Routingverfahren für das Architekturkonzept geeignet ist, damit nach einer entsprechenden Wegewahl eine Kommunikation zwischen Client und Server erfolgen kann.

6.2 Routingverfahren

Um die in Abschnitt 6.1 beschriebenen Funktionen realisieren zu können, müssen die einzelnen Netzwerkkomponenten miteinander kommunizieren können. Die Basis dafür bildet das kontextsensitive Routing selbst. Da es sich hierbei um ein neues Verfahren der Routenfindung handelt, resultiert daraus die Frage, ob ein Protokoll existiert, das möglicherweise den Anforderungen entsprechend modifiziert werden kann. Dazu wird im folgenden Abschnitt darüber diskutiert, welches herkömmliche Routingprotokoll dafür in Frage kommen könnte oder ob es sinnvoller ist, ein komplett neues Routingprotokoll zu entwickeln. In Abschnitt 6.2.3 wird dann erklärt, wie mit Hilfe dieses Protokolls die Kontexttypen und die vom Nutzer initiierte Dienstanfrage übertragen werden. Der Abschnitt 6.2.4 diskutiert schließlich das Verhalten in unterschiedlichen Netzwerkumgebungen.

6.2.1 Auswahl des Basisroutingverfahrens

Um abschätzen zu können, inwieweit ein neues Routingprotokoll entwickelt werden muss, ist eine Analyse der bereits existierenden Protokolle nötig. Diese sind auf ihre Erweiterbarkeit hin zu untersuchen, sodass während einer Routensuche sowohl die Kontexttypen als auch der angeforderte Dienst zum Kontextrouter übertragen werden können. In Kapitel 2 wurde bereits dargelegt, dass gerade Routingprotokolle für Ad-hoc-Netze als geeignet erscheinen und auch heterogene Netzwerkumgebungen unterstützen können. Auf eine Untersuchung von Festnetzprotokollen kann deshalb an dieser Stelle verzichtet werden. Des Weiteren müssen die in Frage kommenden Protokolle den bereits in den Kapiteln 3 und 5 aufgeführten Anforderungen genügen. Da das kontextsensitive Routing auf vorhandene Funktionen bestehender Routingprotokolle aufbaut, ist davon auszugehen, dass auf die Entwicklung eines komplett neuen Routingprotokolls verzichtet werden kann. Unter diesen Voraussetzungen werden die noch verbliebenen Ad-hoc-Netz-Routingprotokolle auf ihre Eignung hin untersucht. Hierbei kann auf Protokolle, die für spezielle Anwendungsbereiche entwickelt wurden (z. B. positionsbasierte Routingverfahren), vorerst verzichtet werden, da ein allgemeines Verfahren gesucht ist. Dieses soll trotzdem netzübergreifend arbeiten, möglichst flexibel einsetzbar und nicht auf bestimmte Anwendungsfälle beschränkt

sein. Zukünftige Forschungsarbeiten und Entwicklungen müssen diese Routingprotokolle jedoch nicht ausschließen, sofern sie die nachfolgenden Ansprüche erfüllen. Bei der Auswahl des geeignetsten Verfahren sind die „Unbedingten Anforderungen“ an Routingprotokolle (siehe Tabelle 3.1 aus Abschnitt 3.3) zu berücksichtigen. Des Weiteren sind folgende Anforderungen von Bedeutung:

Simulator: Da das hier eingeführte kontextsensitive Routing auch mit Hilfe von Simulationen untersucht werden soll, werden bevorzugt Protokolle verwendet, die schon im *network simulator version 2.xx* (ns-2) implementiert sind oder für die bereits Softwarelösungen existieren, die in den ns-2 implementiert werden können. Dadurch ist eine Minimierung des Aufwandes für die Programmierung eines Routingprotokolls möglich, sodass bei einer vorhandenen Implementierung lediglich die Kontexterweiterung in das Protokoll eingebunden werden muss. Detaillierte Informationen zu dieser Thematik finden sich in Kapitel 7.

Verfügbarkeit: Es werden bevorzugt die Protokolle ausgewählt, für die es bereits praktische Implementierungen gibt. Das dabei unterstützte Betriebssystem spielt eine untergeordnete Rolle.

Standardisierung: Die Standardisierung bzw. der Standardisierungsstatus ist insofern wichtig, weil durch diese Information abgeschätzt werden kann, ob das Protokoll von der „Netzgemeinde“ angenommen wurde bzw. werden wird. Dieser Punkt ist für Entwicklungsvorhaben auch deshalb wichtig, weil für nicht standardisierte Protokolle häufig nur unzureichende oder überhaupt keine Dokumentationen bzw. Beschreibungen vorliegen.

Erweiterbarkeit: Soll ein vorhandenes Protokoll als Basis dienen, muss dieses erweiterbar sein. Erweiterbar sind solche Protokolle, die für die Kommunikation Nachrichtenformate verwenden, deren Paketkopf (Header) freie Kapazitäten aufweist. Eine andere Möglichkeit besteht darin, dass die Spezifikation des betrachteten Routingprotokolls eine Erweiterung des gesamten Paketformats erlaubt. Damit wird es möglich, weitere Informationen wie verschiedene Kontexttypen und den angeforderten Dienst zu übertragen. Bei der Beurteilung dieser Anforderung stellen die nicht standardisierten Protokolle das größte Problem dar. Teilweise sind auch die Drafts diesbezüglich nicht sehr aussagekräftig. Deshalb können zum gegenwärtigen Zeitpunkt auch nicht zu jedem Protokoll Aussagen getroffen werden.

Multicast: Wie in Abschnitt 6.3 gezeigt wird, muss das Routingverfahren auch Broadcasts oder Multicasts unterstützen, um die Adresse des Kontextrouters allen Netzknoten mitteilen zu können. In Ad-hoc-Netzen werden zur Adressierung mehrerer oder aller Knoten vorzugsweise Multicasts verwendet. Darüber hinaus unterstützt IPv6 im Gegensatz zu IPv4 nur Multicast. Damit die Funktionen des Architekturkonzepts für beide IP-Versionen kompatibel sind, muss das auszuwählende Protokoll auch Multicasts unterstützen.

Eine Auswertung der Tabelle 3.2 in Kombination mit der in [Pasc05] erstellten sowie für diese Arbeit aktualisierten und erweiterten Tabelle 6.1 führt zu dem Ergebnis, dass gegenwärtig nur AODV, DSR, OLSR und TORA die gestellten Anforderungen

Routing-verfahren	Simulator	Verfügbarkeit	Standardi-sierung	Erweiter-barkeit	Multicast
ABR	nein	k. A.	IETF-Draft	ja	nein
AODV	ja	Windows, Unix	RFC 3561	ja	ja[1]
CBRP	ja	k. A.	IETF-Draft	k. A.	nein
CEDAR	nein	k. A.	IETF-Draft	k. A.	ja[1]
CGSR	nein	k. A.	nein	k. A.	ja[1]
DSDV	ja	k. A.	nein	k. A.	nein
DSR	ja	Windows, Unix	RFC 4728	ja	ja[1]
FSR	ja	k. A.	IETF-Draft	ja	k. A.
LMR	nein	k. A.	nein	k. A.	nein
OLSR	ja	Windows, Unix	RFC 3626	ja	ja[1]
PARO	nein	k. A.	IETF-Draft	k. A.	k. A.
RDMAR	nein	k. A.	IETF-Draft	ja	ja[1]
Spine	nein	k. A.	nein	k. A.	ja[1]
SSR	nein	k. A.	nein	k. A.	nein
STAR	nein	k. A.	IETF-Draft	nein	k. A.
TBRPF	nein	Unix	RFC 3684	ja	ja[1]
TORA	ja	Unix	IETF-Draft	ja	ja[2]
WRP	nein	k. A.	nein	k. A.	ja[1]
ZRP	ja	k. A.	IETF-Draft	ja	ja[1]

Tabelle 6.1: Vergleich der Auswahlkriterien für verschiedene Routingverfahren

erfüllen können. Beim Vergleich der Routingprotokolle ist zunächst auf die zusätzlich in [Pasc05] aufgeführte Anforderung „Effizienz“ verzichtet worden, da die dortigen Aussagen über den Einfluss auf die Verkehrslast subjektiv getroffen wurden und dabei lediglich die Größe des jeweiligen Paketkopfes berücksichtigt wird. Dies reicht aber zur Abschätzung der durch das Routingprotokoll verursachten Verkehrslast nicht aus. Eine Recherche nach vergleichende Messergebnissen blieb erfolglos und der Aufwand eigener Messungen im Rahmen dieser Arbeit war zu hoch, sodass dazu aktuell keine Informationen vorliegen.

AODV, DSR, OLSR und TORA sind im ns-2 implementiert. Außerdem sind sie erweiterbar und unterstützen Multicast. Da für AODV, DSR und OLSR sowohl Implementierungen für die Betriebssysteme Unix-/Linux- als auch für Windows existieren, wurden diese gegenüber TORA favorisiert. Somit wird eine hohe Flexibilität für die praktische Realisierung eines Netzwerkes und bei der Gestaltung von Testszenarien gewährleistet. Des Weiteren haben diese Protokolle bereits den Status eines RFC erreicht. DSR besaß zum Zeitpunkt der Protokollauswahl noch den Status eines IETF-Drafts [JoMH03], sodass es ebenfalls ausschied. Die Veröffentlichung als RFC 4728 [JoHM07] erfolgte erst im Februar 2007. Die finale Entscheidung fiel zugunsten des AODV, da einerseits die zur Verfügung stehenden Implementierungen am geeignetsten erschienen, es sich andererseits um ein reaktives Protokoll handelt (siehe Tabelle 3.2). OLSR ist dagegen ein proaktives Protokoll. In Abschnitt 3.4 wurde bereits auf die unterschiedlichen Arbeitsweisen eingegangen. Da reaktive Pro-

[1] mit Erweiterung bzw. Aufsatz
[2] Voraussetzung: IMEP

tokolle nur auf Anforderung arbeiten, ist AODV für das kontextsensitive Routing effizienter. Eine Routensuche muss hier erst bei einer Dienstanfrage initiiert werden.

Für die Auswahl eines geeigneten Protokolls ist es von wesentlicher Bedeutung, welche Zukunftsaussichten es besitzt. Trotz der großen und stets wachsenden Protokollvielfalt spielt AODV auch heute noch eine bedeutende Rolle. Aktuell wird es z. B. im WLAN-Standard IEEE 802.11s eingesetzt, welcher WLAN-Mesh-Netze spezifiziert [Hier07]. Dies kommt der Interoperabilitätsanforderung gegenüber heterogenen Netzwerkarchitekturen auch beim zukünftigen Einsatz entgegen. Darüber hinaus bestätigt [CoGi07], dass auch noch aktiv an dem Protokoll entwickelt wird. Letztlich erfüllt AODV alle gestellten Anforderungen und soll deshalb als Basis für das kontextsensitive Routing dienen. Die in der Tabelle aufgeführten Einträge „k. A." bedeuten wiederum „keine Angabe". Dieser Fall tritt beispielsweise dann auf, wenn ein Protokoll nicht standardisiert ist und bestimmte Eigenschaften auch durch intensive Recherche nicht be- oder widerlegt werden konnten.

Aus der oben durchgeführten Untersuchung wird ersichtlich, dass auf die Entwicklung eines vollständig neuen kontextsensitiven Routingprotokolls verzichtet werden kann. AODV erfüllt alle Voraussetzungen hinsichtlich einer Modifikation zu einem kontextsensitiven Routingprotokoll. Die Funktionen des in Abschnitt 6.1 dargestellten Konzeptes können damit vollständig umgesetzt werden. Eine Neuentwicklung wird somit nicht als notwendig erachtet, da der Aufwand in keinem Verhältnis zum Nutzen steht. Ein neu entwickeltes Protokoll ist außerdem kein Garant für eine bessere Routingperformance, währenddessen sich AODV bereits in der Praxis bewährt hat.

6.2.2 Funktionsweise des Basisroutingverfahrens

Um die in Abschnitt 6.3 beschriebenen Kommunikationsabläufe nachvollziehen zu können, wird an dieser Stelle zunächst die Funktionsweise des AODV-Protokolls beschrieben. AODV ist ein tabellenbasiertes, reaktives Routingverfahren. Wie bereits in Kapitel 3 erläutert, wird bei einem solchen Routingverfahren erst dann eine Route gesucht, wenn auch Daten zu einem bestimmten Ziel übertragen werden sollen. AODV ist im RFC 3561 [BRPD03] spezifiziert. Es ist schleifenfrei und bestimmt die Routen für eine Unicast-/Multicastkommunikation. Für die Routensuche wird als Transportprotokoll das *User Datagram Protocol* (UDP) verwendet. In den Netzknoten bzw. Routern, die AODV unterstützen, existieren Routingtabellen, die eine bidirektionale Kommunikation ermöglichen. Es wird hierbei zwischen dem eigentlichen Routing und dem Reverse Routing unterschieden. Die Tabelleneinträge für das Routing werden zur Wegewahl zum Zielknoten hin genutzt. Die Einträge für das Reverse Routing dienen dazu, von anderen Knoten erhaltene Antworten (*Route Reply* bzw. RREP) an den Initiator der Wegesuche zurückzusenden. Auch Daten können über solche Routen gesendet werden. In der Routingtabelle werden neben der Zieladresse beispielsweise die Anzahl der Hops bis zum Ziel, das Netzwerkinterface, die Lebensdauer des Eintrags, verschieden Flags (z. B. valid, invalid, ...) usw. verwaltet. Des Weiteren wird noch eine Tabelle angelegt, die die empfangenen Anfragen (*Route Requests* bzw. RREQs) dokumentiert. Durch diese History-Funktion ist es möglich, Duplikate eines RREQ-Paketes zu entdecken und zu verwerfen. Letztlich wird noch eine Liste angelegt, die diejenigen Nachbarknoten enthält, die aktuell kein RREP-Paket empfangen können. Das ist notwendig, da bei mehreren von verschiedenen

Nachbarknoten empfangenen identischen RREQ-Paketen nur das erste ausgewertet wird. Dieses RREQ könnte aber von einem Nachbarknoten initiiert worden sein, der rückwärts keine RREP-Pakete empfangen kann. Damit würde kein Weg zum Ziel gefunden werden, da alternative Routen ignoriert würden. Die in der so genannten *Black List* aufgenommenen Einträge sind nur für eine bestimmte Dauer gültig. Dieses Timeout wird ebenfalls in der Liste vorgehalten.

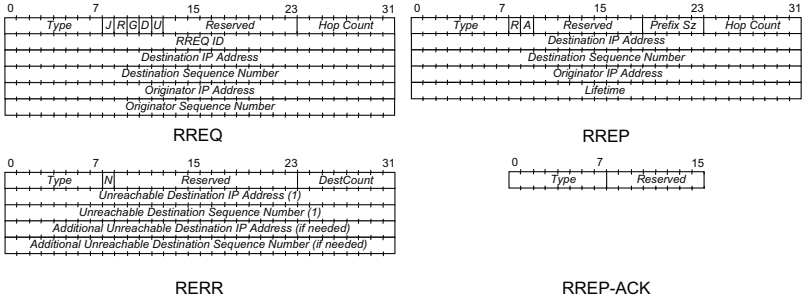


Abbildung 6.2: AODV-Paketformate

Insgesamt werden zum Finden und Managen von Routen vier Nachrichten genutzt. Der Aufbau dieser Nachrichten ist in Abbildung 6.2 dargestellt. Die Nachricht für ein RREQ wird verwendet, um einen Knoten (den Zielknoten) und damit eine Route zu diesem zu finden. Mit einem RREP erfolgt die Antwort, sobald der Zielknoten bzw. der Weg zum Zielknoten gefunden wurde. Wird mit dem RREP eine Bestätigung für dessen Empfang gefordert (A-Flag gesetzt), erfolgt dies mit einem *Route Reply Acknowledgement* (RREP ACK). Schließlich wird ein *Route Error* (RERR) gesendet, wenn bestehende Routen unterbrochen wurden. Die Bedeutung der in den Nachrichtenformaten verwendeten Felder werden im Folgenden kurz erläutert:

Type - beschreibt den Typ des Nachrichtenformats („1“ = RREQ; „2“ = RREP; „3“ = RERR; „4“ = RREP-ACK).

J - *Join*-Flag ist reserviert für Multicast.

R - *Repair*-Flag ist reserviert für Multicast.

G - *Gratuitous*-Flag kennzeichnet, dass ein Zwischenknoten ein so genanntes Gratuitous-RREP zum im Feld „Destination IP Address“ angegebenen Knoten senden muss, sofern dieser Zwischenknoten direkt auf das RREQ antwortet.

D - *Destination Only*-Flag kennzeichnet, dass nur der Zielknoten auf dieses RREQ antworten soll.

U - *Unknown Sequence Number*-Flag wird gesetzt, wenn die Sequenznummer für den Zielknoten unbekannt ist, d. h., wenn z. B. noch keine Route zu dem Zielknoten existiert.

A - *Acknowledgment*-Flag wird gesetzt, wenn die Gefahr besteht, dass die Übertragungsstrecke fehleranfällig oder unidirektional ist. Der Empfänger des RREP muss daraufhin ein RREP-Ack zurück senden.

N - *No Delete*-Flag wird gesetzt, wenn ein Knoten die Wiederherstellung eines lokalen Links initialisiert hat und somit die Route in den anderen Knoten in Richtung Upstream nicht gelöscht werden soll.

Reserved - reservierter und damit ungenutzter Bereich.

Hop Count - Anzahl der Hops vom Initiatorknoten bis zu dem Knoten, der die Anfrage bearbeitet.

RREQ ID - kennzeichnet zusammen mit der IP-Adresse des Initiatorknotens ein RREQ eindeutig.

Destination IP Address - ist die IP-Adresse des Zielknotens.

Destination Sequence Number - ist die aktuelle bzw. zuletzt bekannte Squenznummer vom Zielknoten.

Originator IP Address - ist die IP-Adresse des Knotens, der das RREQ-Paket initiiert hat.

Originator Sequence Number - ist die aktuelle Sequenznummer des Knotens, der das RREQ initiiert hat.

Prefix Sz - sofern der Wert (*Prefix Size*) nicht Null ist, gibt er das Subnetz an. So gekennzeichnete Routen können dann für jeden Knoten mit der gleichen Subnetzmaske wie der eigentliche Zielknoten verwendet werden.

Lifetime - ist die Zeit in Millisekunden nach dem Empfang des RREP-Paketes, für die eine Route als valid gilt.

DestCount - ist die Anzahl nicht erreichbarer Ziele, die im RERR-Paket enthalten sind, und muss deshalb mindestens 1 betragen.

Unreacheable Destination IP Address (1) - ist die IP-Adresse des nicht erreichbaren Knotens. Weitere IP-Adressen können ebenfalls angegeben werden (Feld: *Additional Unreachable Destination Sequence Numbers (if needed)*).

Unreachable Destination Sequence Number (1) - ist die Sequenznummer, die in der Routingtabelle dem nicht erreichbaren Zielknoten zugeordnet ist. Sofern weitere IP-Adressen nicht erreichbarer Knoten angegeben werden, sind auch die zugehörigen Sequenznummern zu übertragen (Feld: *Additional Unreachable Destination IP Addresses (if needed)*).

AODV zeichnet sich durch einen kleinen Overhead aus, was die Verkehrslast positiv beeinflusst. Die Funktionsweise von AODV soll anhand einer Routensuche allgemein erläutert werden. Fehler- und Ausnahmesituationen werden dabei vernachlässigt. Für eine detaillierte Beschreibung wird auf den RFC 3561 verwiesen. In Abbildung 6.3 ist ein Beispiel für eine Routensuche dargestellt. Knoten A möchte hier Daten zu Knoten I übertragen. Dazu wird ein RREQ an alle Nachbarn gesendet. Direkt erreichbare Nachbarn sind in der Abbildung mit einer Linie verbunden. Diese leiten die Anfrage wiederum an ihre Nachbarn weiter, sofern der Zielknoten nicht bekannt ist oder das D-Flag gesetzt ist. Duplikate werden dabei erkannt und gelöscht. Jeder

Knoten der ein RREQ erhält, speichert die Routeninformation zum rückwärtigen Erreichen des Initiators der Anfrage ab. Wurde der Zielknoten erreicht (Darstellung ⓓ in Abbildung 6.3), sendet dieser ein RREP an den Initiator (Knoten A) zurück. Das RREP findet den Weg zur Quelle über die Reverse-Route-Einträge der Knoten. Gleichzeitig aktualisiert jeder Knoten, der das RREP weiterleitet, seine Routingtabelle und kennt damit auch den Weg zum Knoten I.

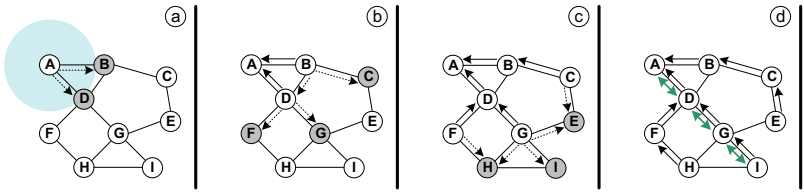


Abbildung 6.3: AODV-Routingalgorithmus [Pasc05]

Ergänzend sei noch erwähnt, dass der RFC 3561 AODV basierend auf IPv4 beschreibt. Mittlerweile befindet sich auch das „*Ad hoc On-Demand Distance Vector (AODV) Routing for IP version 6*“ [PeRD00] in der Standardisierungsphase. Prinzipiell arbeitet dort das Routingverfahren genauso wie unter IPv4. Die Paketformate wurden dahingehend erweitert, dass nun 128-Bit-lange Quell- und Ziel-IP-Adressen (*Source IP Address, Destination IP Address*) verwendet werden. Beim RREQ wird des Weiteren auf die Flags D und U verzichtet. Damit erweitert sich der reservierte Bereich um 2 Bit. Die *RREQ ID* wird durch die gleichlange *Flooded Packet ID* ersetzt. Da IPv6 keine Broadcast-Adressen kennt, wird zur Routensuche mit Multicast gearbeitet. Hierzu ist in der Spezifikation eine Multicast-Adresse definiert, die im aktuellen Subnetz wie ein Broadcast funktioniert, sodass alle Knoten des Subnetzes erreicht werden können. Nähere Details zur Konfiguration des AODV unter IPv6 finden sich in [PeRD00].

6.2.3 Erweiterung des Basisroutingverfahrens

Das in dieser Arbeit vorgestellte Konzept sieht nur die Übertragung von Kontexttypen mit Hilfe des AODV vor. Die Übertragung der zu den Kontexttypen zugehörigen Werte wird nach Abschnitt 5.3.1 als wenig sinnvoll erachtet. Die Anfrage mittels RREQ-Paket dient lediglich dazu einen passenden Server zu finden, der den angeforderten Dienst erbringen bzw. die angegebenen Kontexttypen verarbeiten kann. Im Anschluss an die Wegewahl werden die Kontextinformationen direkt zwischen Anwendung und Dienst bzw. Client und kontextsensitivem Server ausgetauscht. Der Wegewahlprozess wird diesbezüglich somit entlastet.

Zur Erweiterung bzw. Modifikation der AODV-Pakete stehen zwei Varianten zur Verfügung:

1. Verwendung nicht genutzter (reservierter) Felder im Paketformat
2. Erweiterung des Paketes um zusätzliche Felder

Für die Nutzung der ersten Variante stehen in verschiedenen Nachrichtenformaten Felder zur Verfügung, die in Abbildung 6.2 durch „*Reserved*“ gekennzeichnet sind.

Reservierte Bereiche gibt es für RREQ- (11 Bit bei IPv4, 13 Bit bei IPv6), RREP- (9 Bit bei IPv4, 7 Bit bei IPv6) und RERR-Pakete (15 Bit für IPv4 und IPv6). Da der Teilnehmer bei einer Dienstsuche als erstes eine entsprechende Anfrage an das Netz sendet, sind an dieser Stelle auch die Kontexttypen und der gewünschte Dienst mit zu übertragen. Ein Netzknoten, der diese Anfrage dann auswerten kann (also in der Architektur der Kontextrouter), liest die gesuchten Kontexttypen aus und verarbeitet diese entsprechend. Aus dieser Funktionsweise heraus ergibt sich, dass lediglich das RREQ-Paket zur Übermittlung dieser zusätzlichen Informationen benötigt wird. Mit diesem Paket stehen elf freie Bits zur Verfügung. D. h., es können theoretisch bis $2^{11} - 1 = 2047$ verschiedene Kontexttypen unterstützt werden. Aktuelle Entwicklungen und Forschungsarbeiten (z. B. [LeDS05, DeLS05, SFB06, Lewa07]) zeigen aber, dass kontextsensitive Anwendungen mehrere Kontexttypen gleichzeitig berücksichtigen und benötigen. Um beliebige Kombinationen von diesen Typen mittels RREQ übertragen zu können, müsste pro Kontexttyp ein Bit reserviert werden. Das bedeutet jedoch, dass nur 11 verschiedene Kontexttypen verwendet werden können. Wie aber schon in Kapitel 4 gezeigt wurde, reicht diese Anzahl bei weitem nicht aus. Hinzu kommt, dass mit der Weiterentwicklung der Technik eine immer genauer werdende Beschreibung des Kontextes möglich wird. Kontexttypen, die heute vielleicht messtechnisch noch nicht erfassbar sind (z. B. der mentale Zustand eines Individiums), können dann aufgenommen und ausgewertet werden. Diese Eigenschaft wird die Zahl der verwertbaren Kontexttypen auch zukünftig noch ansteigen lassen.

Eine bessere Alternative gegenüber der Nutzung reservierter Bits bietet die zweite Variante einer Modifikation. Hierfür unterstützt AODV nach RFC 3561 die Erweiterung der Formate von RREQ- und RREP-Paketen. Die Nutzung dieser Erweiterungsmöglichkeiten wird auch in [KoPe02] vorgeschlagen. Allerdings sind dort wie auch beim davon abgeleiteten AODV-SD keine Möglichkeiten in den Paketerweiterungen vorgesehen um Kontexttypen zu übertragen, womit eine Nutzung innerhalb der kontextsensitiven Routingarchitektur ausscheidet. Weitere Defizite dieses Protokolls wurden bereits in Abschnitt 4.6.2 erörtert.

Mit den RREQ- und RREP-Paketen können jeweils bis zu 255 Byte an zusätzlichen Daten übertragen werden. Dazu wird das Paketformat um die in Abbildung 6.4 dargestellten Anteile erweitert. Mit dem Feld *Type* kann eine Kategorisierung der angehängten Daten erfolgen. Der Wert 1 ist hierbei schon für das *Hello Interval Extension*-Format vergeben. Das *Length*-Feld gibt die Anzahl der zu übertragenden Daten in Bytes an. Das Feld *type-specific data* enthält die eigentlichen Daten.

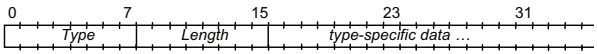


Abbildung 6.4: Erweiterung für RREQ- und RREP-Pakete

Mit den in den Erweiterungen der RREQ- und RREP-Paketen zur Verfügung stehenden 255 Bytes können theoretisch bis zu $255 * 8 - 1 = 2039$ verschiedene Kontexttypen gleichzeitig übertragen werden. Zusätzlich müssen auch noch der gewünschte Dienst und eventuell zusätzliche Informationen berücksichtigt werden, sodass sich diese Zahl entsprechend verringert. Trotzdem bietet diese Möglichkeit auch in absehbarer Zukunft genügend Reserven. Zu beachten ist jedoch, dass sich die Größe der RREQ-/RREP-Pakete bei Verwendung aller Bytes um etwa 11 mal vergrößert. Eine Effizienzsteigerung diesbezüglich könnte dann durch die variable Nutzung des

Längenfelds erfolgen, weil damit die Größe des Datenfelds an die aktuell zu übertragenden Kontexttypen angepasst werden kann. Neben diesen Feststellungen ist zu diskutieren, inwiefern es sinnvoll ist, die Daten strukturiert zu übertragen. Vorschläge dazu werden in Abschnitt 6.3.2.4 unterbreitet.

Damit das kontextsensitive Routing mit Hilfe von erweiterten RREQ auch tatsächlich funktioniert, ist im Paketkopf das D-Flag per Default zu setzen. Der Grund liegt in der Arbeitsweise des AODV. Empfängt ein Zwischenknoten ein RREQ, überprüft dieser in seiner Routingtabelle, ob ihm die Route zum Ziel schon bekannt ist. Ist dies nicht der Fall, erfolgt ein Broadcast an alle benachbarten Knoten. Kennt der Zwischenknoten jedoch schon eine Route zum Ziel, antwortet er sofort mit einem RREP. Mit dem Setzen des D-Flags wird nun sichergestellt, dass das RREQ tatsächlich zum Ziel weitergeleitet wird. Die Nutzung des D-Flags im Konzept wirkt sich auch auf die Kompatibilität bei Verwendung des IPv6 aus, da es in dem zugehörigen *Internet Draft* [PeRD00] nicht vorgesehen ist. Vielmehr werden dort zusätzlich modifizierte RREQ- und RREP-Paketformate angeboten, mit denen 255 Bytes Nutzdaten direkt zum Ziel gesendet werden können. Bezeichnet werden diese als Optionen/Alternativen. Eine Anfrage erfolgt dann mit dem *Flood Data Option*-Format und eine Antwort mit dem *Flood Data Reply Option*-Format.

Werden die Erweiterungen der RREQ-/RREP-Pakete, wie in diesem Abschnitt beschrieben, genutzt, hat das keine Auswirkungen auf solche AODV-Knoten, die keine Kontextdaten auswerten können. Die Erweiterung wird ignoriert und das Paket als solches weitergeleitet. Ein konkretes Beispiel für die erweiterten RREQ-/RREP-Pakete gibt Abschnitt 6.3.4.

6.2.4 Verhalten in heterogenen Netzwerkumgebungen

Um abschätzen zu können, wie sich das kontextsensitive Routingverfahren in heterogenen Netzen verhalten wird, muss dessen Protokollstruktur näher beleuchtet werden. Hierbei unterscheidet sich das AODV nicht von der modifizierten Variante aus der konzipierten Routingarchitektur. Nach RFC 3561 setzt das Routingprotokoll auf UDP auf. UDP ist ein verbindungsloses Transportprotokoll, was wiederum IP auf der Vermittlungsschicht nutzt.

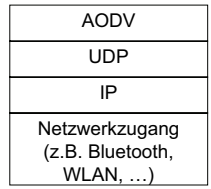
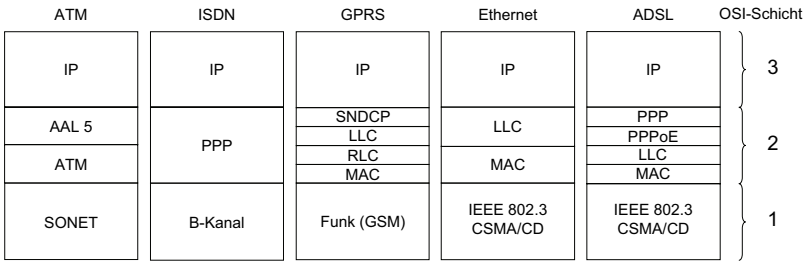


Abbildung 6.5: Protokollstack für AODV

Abbildung 6.5 zeigt den vollständigen Protokollstack. Aus dieser Darstellung lassen sich bereits einige Eigenschaften bezüglich des Verhaltens in heterogenen Netzen ableiten. Z. B. spielt die verwendete Netzwerktechnik keine Rolle, sofern diese das IP unterstützt. Mittlerweile können das sehr viele Netze, unabhängig davon, ob diese für den Sprach- oder Datenverkehr konzipiert wurden. Stellvertretend seien hier ATM (*Asynchronous Transfer Mode*), POTS (*Plain Old Telephone Service*),

ISDN, GSM, GPRS, Ethernet, WLAN, Bluetooth etc. genannt. Auch Teilnehmerzugangstechniken wie sämtliche Varianten von DSL erfüllen diese Voraussetzungen. In Abbildung 6.6 sind stellvertretend einige Beispiele für mögliche IP-Protokollstacks aufzeigt.



Bedeutung: AAL 5 (*ATM Adaptation Layer 5*); LLC (*Logical Link Control*); PPP (*Point-to-Point Protocol*); PPPoE (*PPP over Ethernet*); RLC (*Radio Link Control*); SNDCP (*Subnetwork Dependent Convergence Protocol*); SONET (*Synchronous Optical Network*)

Abbildung 6.6: IP-Protokollstacks für verschiedene Übertragungstechniken

Nach RFC 3561 wird für AODV der UDP-Port 654 verwendet. Für das IP-Paket (IPv4) ist die Broadcast-Adresse (255.255.255.255) zu setzen. Es handelt sich um ein *Limited* Broadcast, womit die AODV-Pakete im Ad-hoc-Netz weitergeleitet werden. Im IP-Infrastrukturnetz wird ein solches Broadcast zumindest an alle Knoten im eigenen IP-Subnetz gesendet. Damit ist die korrekte Funktion innerhalb des Subnetzes unabhängig von der verwendeten Netztechnik gewährleistet. Problematisch wird die globale Kommunikation über IP-Subnetzgrenzen hinweg. IP-Router, die Netze oder Subnetze miteinander verbinden, leiten in der Regel keine Broadcasts oder Multicasts weiter. Auf diese Funktionalitäten wird aus Sicherheitsgründen verzichtet. Die Weiterleitung von *Limited* Broadcasts ist dabei strikt verboten. Im RFC 2644 [Seni99] ist für Router darüber hinaus explizit vorgeschrieben, dass diese in der Grundkonfiguration keine *Directed* Broadcasts weiterleiten dürfen. *Directed* Broadcasts werden an alle Knoten eines bestimmten Netzes oder Subnetzes gesendet. Andererseits entscheiden die Betreiber der Netze letztlich selber, inwieweit sie Broadcasts und Multicasts in die Subnetze unterstützen. AODV-RREQs verbleiben damit jedoch per Default in den ursprünglichen Subnetzen, d. h. in den Netzen, in denen sie initiiert wurden. Auch bei IPv6 bleibt dieses Ergebnis unverändert, da diese Version keine Broadcasts unterstützt. Dafür wird aber eine Multicast-Adresse bereitgestellt, mit der alle Knoten eines Subnetzes erreichbar sind.

Die Auswirkungen auf die in Abschnitt 1.2 beschriebene Zielsetzung der Arbeit bleiben jedoch aus folgendem Grund minimal. Heutige kontextsensitive Dienste stehen meist unmittelbar in Bezug zur Position des Nutzers. Daraus ergibt sich, dass der Nutzer vorrangig Dienste abrufen wird, die lokal, d. h. in seinem Umfeld, angeboten werden. Dadurch lohnt es sich wiederum mindestens einen kontextsensitiven Router innerhalb solcher Netze bzw. administrativer Domänen zu betreiben. So werden beispielsweise bei der Suche nach bestimmten Restaurants, Geschäften etc. geeignete Wege zu einem Zielpunkt erwartet, die sich im Umfeld des Nutzers befinden. Beschreibt der Kontext beispielsweise den Gesundheitszustand des Nutzers, ist es bei einem kritischen Zustand am sinnvollsten, vom Endgerät automatisch

die nächstgelegene geeignete Hilfe zu rufen. Nutzer, Dienst und Position stehen also bei vielen kontextsensitiven Diensten in unmittelbarer Beziehung. Sicherlich lassen sich hier noch weitere Beispiele finden. Deutlich wird jedoch, dass die kontextsensitiven Router in solchen Netzen platziert werden müssen, in denen eine Vielzahl von kontextsensitiven Diensten direkt abgerufen werden sollen. Dadurch kann auf eine RREQ-Anfrage über Subnetzgrenzen hinweg verzichtet werden. Nicht benötigte AODV-Funktionen (z. B. Hello-Pakete) sollten von den betreffenden Netzknoten allerdings in IP-Infrastrukturnetzen vermieden werden, da sich diese negativ auf die Verkehrsbilanz auswirken.

Werden heterogene Netzwerke betrachtet, ist dabei zu berücksichtigen, dass die Nutzer und damit die Clients mobil sein können. Deshalb erfolgen dort auch notwendigerweise Handover. Das können sowohl vertikale als auch horizontale Handover sein. Auf diese Problematik wurde bereits in Abschnitt 3.2 eingegangen. Der Verlust einer Route führt im Ad-hoc-Netz zum Versenden eines RREQ, d. h. zu einer neuen Routensuche und damit zum horizontalen Handover. Komplexer gestaltet sich die Situation, sobald ein vertikaler Handover zwischen verschiedenen Netztechniken erfolgen soll. Erste Untersuchungen, wie zwischen diesen Handover erfolgen können, werden in [Pein04] näher beleuchtet. Eine praktische Implementierung beschreibt [Ever04], bei der bereits ein Handover zwischen GPRS und WLAN realisiert wurde. Dort wurde auch gezeigt, dass ein nahtloser Handover möglich ist und somit ein ständiger Zugang zum IP-Netz erfolgen kann. Die Funktionalität des kontextsensitiven Routings wird also weiterhin vollständig unterstützt.

Daneben müssen auch so genannte Server-Handover bzw. ein Rerouting unterstützt werden. Die Notwendigkeit dazu ergibt sich beispielsweise, wenn ein Server ausfällt oder die Kommunikation zu einem Client wegen Überlastung abgebrochen wurde. Steht in diesem Fall ein alternativer Server zur Verfügung, so sollte ein Rerouting dorthin erfolgen. Im Idealfall bemerkt der Nutzer nichts davon. In Anbetracht der im Architekturkonzept vorgeschlagenen Proxykommunikation können die dazu notwendigen Funktionalitäten nur vom Kontextrouter erbracht werden. Was dabei zu berücksichtigen ist, wird in Abschnitt 6.3.2.6 näher beschrieben.

6.3 Kommunikationsprozesse

Dieses Kapitel geht detailliert auf die Kommunikationsabläufe zwischen den kontextsensitiven Komponenten ein. Es wird dabei beschrieben, welche Aufgaben die kontextsensitiven Server, Kontextrouter und Clients übernehmen und welche Voraussetzungen sie erfüllen müssen. Grundlage dafür bildet im folgenden Kapitel die Diskussion darüber, wie die Verbreitung der zu den Kontextroutern gehörenden Adressinformationen im Netzwerk erfolgen soll.

6.3.1 Adressierung

Bei dem in diesem Kapitel vorgeschlagenen Architekturkonzept basiert die Kommunikation auf der Nutzung von IP. Dementsprechend wird die Adressierung aus diesem Protokoll verwendet. Damit die Netzknoten sich untereinander erreichen können, müssen diese die Adressen der jeweiligen Kommunikationspartner kennen. Die zentrale Komponente bildet hier der Kontextrouter. Um mit diesem kommunizieren zu können, müssen entweder die Server und Clients die IP-Adresse des Routers kennen oder aber der Router kennt die IP-Adressen potentiell zu erreichender

Netzknoten. Aus verkehrstheoretischen und funktionalen Erwägungen heraus ist es sinnvoller, dass die Routeradresse den kontextsensitiven Servern und Clients bereitgestellt wird. Zum einen fragen die Clients bei Bedarf nach einem kontextsensitiven Dienst, zum anderen müssen sich die Server beim Kontextrouter registrieren. Beide Knotentypen sind damit Initiator einer Kommunikation und benötigen deshalb die Router-IP-Adresse. Die Initiierung der Kommunikation geht also immer von den Clients und Servern aus. Des Weiteren können sich die kontextsensitiven Server, wie in Abschnitt 6.2.4 beschrieben, in anderen Subnetzen befinden. Für den Kontextrouter wären sie dann nicht „sichtbar“. Wird dabei berücksichtigt, dass für eine Dienstanfrage an den Router ein erweitertes AODV-RREQ versendet werden soll, ergeben sich die im Folgenden beschriebenen Kommunikationsmöglichkeiten.

6.3.1.1 Möglichkeiten der Adressierung

Verschiedene Möglichkeiten bezüglich der Adressierung des Kontextrouters wurden bereits in [Schm05] diskutiert. Die Ergebnisse dienten als Ausgangspunkt, wurden aber für die folgende Gegenüberstellung weiterentwickelt. Dabei wird davon ausgegangen, dass die jeweils genannte Kommunikationsart vom Client bzw. Server lediglich dazu verwendet wird, um den Kontextrouter direkt erreichen zu können.

Unicast: Hierbei handelt es sich um eine Punkt-zu-Punkt-Kommunikation. Das bedeutet für den Client bzw. Server, dass er bei einer Kommunikation mit dem Kontextrouter dessen IP-Adresse kennen muss. Um diese Adresse im Netzwerk bekannt zu machen, stehen folgende Varianten zur Verfügung:

- **Variante 1:** Es werden Adressen festgelegt, die für jedes Netz gelten. Diese so genannten Well-Known-Adressen sind dann reserviert und dürfen in jedem Netz nur für die Kontextrouter zur Verfügung stehen.
- **Variante 2:** Der Nutzer muss sich beim Systemadministrator erkundigen, welche Kontextrouter für ihn in Frage kommen. Perspektivisch könnte dies für Netzwerke, die mit dem DHCP arbeiten, automatisiert werden. Dazu müsste aber das DHCP um diese Möglichkeit erweitert werden.
- **Variante 3:** Zur Verbreitung ihrer IP-Adressinformationen nutzen die Kontextrouter so genannte Leuchtfeuer (Beacons). Der Router sendet diese Informationen periodisch in kurzen zeitlichen Abständen via Broadcast in das Netzwerk.
- **Variante 4:** Der Router sendet periodisch so genannte Advertisement-Pakete in das Netzwerk. Dadurch ist jedem Netzknoten bekannt, wie die Adresse des Routers lautet. Das Advertisement-Paket wird per Broadcast bzw. Multicast übertragen. Im Gegensatz zu Beacons ist der zeitliche Abstand zwischen zwei gesendeten Advertisements wesentlich größer. Die Knoten können deshalb die Adressinformationen auch mit Hilfe von Solicitation-Paketen anfordern.

Anycast: Eine Erklärung des Anycast-Mechanismus erfolgte bereits in Abschnitt 5.2. Bei dieser Art der Kommunikation wird eine Anfrage in das Netzwerk gesendet. Der erste Netzknoten, der sich für diese Anfrage verantwortlich

zeichnet, antwortet. Ein solcher Ansatz kann auch zum Auffinden der Kontextrouter verwendet werden. Danach würde ein AODV-RREQ an alle Netzknoten einer administrativen Domäne gesendet werden. Der erste Kontextrouter würde auf diese Anfrage antworten. Allerdings setzt eine Adressierung mit Hilfe von Anycasts voraus, dass alle Kontextrouter neben ihrer eindeutigen Unicast-Adresse auch eine gemeinsame Anycast-Adresse besitzen. Dies würde die Komplexität der Architektur erhöhen, da zusätzliche Installationen auf den Geräten, die Anycast unterstützen sollen, notwendig sind. Des Weiteren wäre keine dedizierte Auswahl eines alternativen Kontextrouters möglich.

Multicast: Bei der Nutzung von Multicasts würden die Kontextrouter eine spezielle Multicast-Adresse erhalten. Hierbei handelt es sich bei IPv4 um eine Adresse aus dem Klasse-D-Netz (224.0.0.0 - 239.255.255.255). Bei IPv6 würde dies eine Adresse aus dem Bereich ff00::/8 sein. Die Nutzung wird durch die *Internet Assigned Numbers Authority* (IANA) [MIPv08, MIPv07] reguliert. Bei einer Kommunikation über eine Multicast-Adresse würden die Daten an alle erreichbaren Kontextrouter gesendet werden, die dann auch entsprechend darauf reagieren müssten.

Concast: Beim Concast gibt es mehrere Sender und einen Empfänger. Im hier betrachteten Szenario tritt dieser Fall ein, wenn mehrere Nutzer parallel eine Anfrage an einen Kontextrouter senden würden. Damit ist dies ein Spezielfall des Unicast und es gelten die dort getroffenen Aussagen.

Broadcast: Bei einem Broadcast sendet ein Knoten an alle anderen Knoten des Netzes. Dadurch würde bei dem hier betrachteten Konzept das gesamte Netz mit der Anfrage nach einem Kontextrouter „geflutet“. Zur gezielten Adressierung bestimmter Netzknoten ist diese Methode nicht geeignet.

6.3.1.2 Gegenüberstellung

Adresstyp	Adressbezug	Initiator	Art der Adresse	Reichweite	Proxykomm.	alternative Auswahl
Unicast	manuell	—	well-known	+	+	+
			individuell	+	+	+
	Beacons	Router	individuell	–	+	+
	Advertisement	Router	individuell	–	+	+
	Solicitation	Clients; Server	individuell	–	+	+
Anycast	manuell	—	well-known	+	o	–
Multicast	manuell	—	well-known	–	–	–

Bedeutung: + → ja, o → eingeschränkt, – → nein

Tabelle 6.2: Gegenüberstellung der Adressierungsmöglichkeiten

Die vorgestellten Adressierungsmöglichkeiten besitzen verschiedene Eigenschaften, die in Tabelle 6.2 gegenübergestellt sind. Broadcasts werden dabei aus den oben genannten Gründen nicht mit betrachtet. Zu jedem vom Kontextrouter unterstützten „Adresstyp“ wird aufgeführt, wie die Clients und Server dessen IP-Adresse erhalten können (Spalte: „Adressbezug“) und welcher der Netzknoten die Verbreitung der

Routeradresse initiiert (Spalte: „Initiator“). Dafür gibt es drei Möglichkeiten. Die IP-Adresse wird manuell vom Nutzer oder einem Administrator übergeben, die Kontextrouter senden ihre Adresse in das Netzwerk (Advertisement, Beacons) oder die Knoten suchen selbständig nach Kontextroutern (Solicitations). In der Spalte „Art der Adresse“ wird angegeben, welche IP-Adressen für die Kontextrouter verwendet werden. Well-known-Adressen sind hierbei durch ihre statische Zuordnung als unflexibel zu bewerten. Im Gegensatz dazu werden indiviuell konfigurierbare Adressen als vorteilhaft betrachtet. Weiterhin ist in der Spalte „Reichweite“ für die jeweiligen Adresstypen aufgeführt, ob diese das Auffinden von Kontextroutern über die Grenzen einer administrativen Domäne hinaus unterstützen. Die folgende Spalte „Proxykomm.“ trifft eine Aussage darüber, ob die im Architekturkonzept vorgeschlagene Proxykommunikation realisiert werden kann. Hierbei vermittelt der Kontextrouter die Daten bei der Kommunikation zwischen Client und Server (siehe Abschnitt 6.1). Dazu muss die IP-Adresse des Kontextrouters innerhalb einer administrativen Domäne jedoch eindeutig sein. Die Spalte „Alternative Auswahl“ beschreibt, ob mit dieser Adressierung ein individueller Zugriff auf verschiedene Kontextrouter möglich ist.

6.3.1.3 Bewertung

Die einzelnen Adressierungsvarianten werden im Folgenden weiter untersucht, um eine für das kontextsensitive Routing geeignete Methode zu finden.

Multicasts unterstützen theoretisch eine Kommunikation über Subnetzgrenzen hinweg. In der Praxis wird diese Funktion von den Routern in IP-Infrastrukturnetzen aus Sicherheitsgründen nicht realisiert bzw. ist bei diesen deaktiviert. Deshalb können nur Kontextrouter innerhalb einer administrativen Domäne gefunden werden. Die Adresse des Kontextrouters ist den Clients/Servern manuell zu übergeben. Die Proxykommunikation ist bei Multicastadressierung nur dann realisierbar, wenn sich nur ein einziger Kontextrouter im Subnetz befindet. Im Ad-hoc-Netz ist jedoch ein Angebot alternativer Kontextrouter erwünscht. Eine Auswahl einzelner Kontextrouter durch Client/Server ist damit ebenfalls nicht möglich.

Anycast ist innerhalb einer administrativen Domäne nur mit erhöhtem Aufwand umsetzbar, da den Routern neben der Anycast-Adresse auch individuelle Adressen zugewiesen werden müssen. Erst in diesem Fall kann eine Proxykommunikation unterstützt werden. Diese Methode ist, wie bereits in Abschnitt 5.2.1.2 beschrieben, prototypisch für WLANs umgesetzt worden. Allerdings würde dann die Kommunikation nach dem Auffinden des Kontextrouters wieder über eine Unicast-Adressierung erfolgen. Eine Kommunikation über Subnetzgrenzen hinweg ist möglich. Der Zugriff auf einzelne individuell ausgewählte Kontextrouter ist nach dem Anycast-Ansatz nicht möglich, da immer der erste als Kontextrouter erkannte Netzknoten verwendet wird.

Alle Unicast-Varianten unterstützen sowohl eine Proxykommunikation als auch den individuellen Zugriff auf einzelne Kontextrouter. Im Detail führt ein Vergleich zu folgenden Ergebnissen:

Variante 1: Werden Well-Known-Adressen verwendet, müssen diese für jeden einzelnen Kontextrouter festgelegt und deren Gültigkeit im Netzwerk garantiert werden. Das muss ggf. über eine Standardisierung erfolgen. Darüber hinaus ist

dies für IPv4 im Infrastrukturnetz aufgrund der Strukturierung der Adressbereiche nur schwer umsetzbar. Bei IPv6 ist dies ähnlich, da es auch hier unterschiedliche Subnetze gibt. Die Vergabe eines von den Netzwerken unabhängigen IP-Adresspools entfällt, da der Kontextrouter eine IP-Adresse aus dem eigenen IP-Subnetz benötigt. Eine Lösung dagegen wäre, einen endlichen Pool von IP-Adressen aus verschiedenen Netzwerken bzw. Subnetzen bereitzustellen, um dann die einzelnen Netze bedienen zu können. Diese Lösung ist jedoch sehr unflexibel, da die Anzahl der Kontextrouter begrenzt und auf bestimmte Netze reduziert wäre. Daneben bestünde auch die Möglichkeit ein IP-Adressschema zu verwenden, das dem von herkömmlichen Routeradressen angelehnt ist. Es könnte z. B. festgelegt werden, dass die größte IP-Adresse des Subnetzes, um 2 minimiert, verwendet wird. In diesem Fall, besteht aber einerseits die Unsicherheit, dass diese Adresse schon vergeben ist, da sich auch mehrere Kontextrouter in einem Netz befinden können. Andererseits arbeitet jeder Netzknoten in einem Ad-hoc-Netz auch als Router, sodass dort dieses Adressschema nicht verwendet werden kann.

Variante 2: Auch die zweite Unicast-Variante, bei der zum Bezug der Routeradresse erst eine Anfrage beim Systemadministrator erfolgen muss, gestaltet sich als unflexibel. Die Adressen müssen manuell bei den Clients und Servern eingetragen werden. Eine Realisierung per DHCP erscheint hier als sehr vielversprechend, hat aber zwei entscheidende Nachteile. Zum einen besitzt DHCP keine vollständige Netzdurchdringung. D. h., viele Netze arbeiten mit statischen Konfigurationen. Zum anderen müsste die Spezifikation des DHCP erweitert werden, damit es die Clients mit zusätzlichen Informationen über Kontextrouter versorgen kann. Eine diesbezügliche Änderung der Spezifikation wird mit Sicherheit aber erst dann erfolgen, wenn sich das kontextsensitive Routing in der Praxis bewährt hat.

Variante 3: Für das vorgeschlagene Architekturkonzept ist die Nutzung von Beacons ungeeignet, weil sich die Verkehrslast durch die kurzen Zeitabstände für das Versenden der Daten stark erhöht. Mit jedem weiteren Kontextrouter innerhalb einer administrativen Domäne wird der durch Beacons verursachte Verkehr vervielfacht. Eine Verbreitung der IP-Adresse ist auch mit dieser Variante nur innerhalb einer administrativen Domäne möglich.

Variante 4: Bei dieser Variante verbreitet ein Kontextrouter die Informationen über seine Anwesenheit und seine IP-Adresse selbst. In Verbindung mit Solicitation können die Abstände zwischen einzelnen Advertisements relativ groß gewählt werden. Deshalb ist die Bilanz für die Verkehrslast wesentlich besser als bei der Verwendung von Beacons. Die Verbreitung der Adresse des Kontextrouters erfolgt nur innerhalb der administrativen Domäne. Dabei ist jedoch zu berücksichtigen, dass Router auch als Verbindungsglied zwischen verschiedenen Netzen bzw. Subnetzen dienen können. Advertisements werden in diesem Fall in alle angeschlossenen Netze gesendet. Eine Weiterleitung über die Grenzen dieser administrativen Domänen hinaus ist aber in einem Infrastrukturnetz in der Regel nicht möglich. Das Gleiche gilt für Solicitations. Eine größere Reichweite kann beispielsweise über den Einsatz von Relay-Agenten oder Proxys erreicht werden. Leiten diese nur die Advertisements und Solicitations weiter, können andere Multicast-/Broadcast-Daten trotzdem blockiert werden. Eine

Realisierung ist jedoch nur mit entsprechender Unterstützung der Netzwerkprovider möglich.

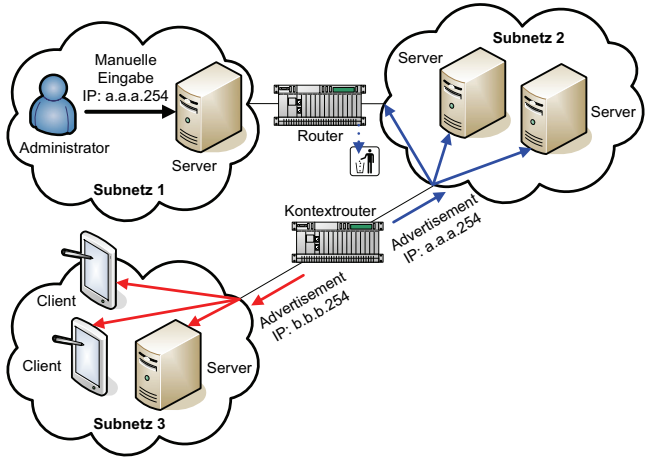


Abbildung 6.7: Die Verbreitung der Adresse des Kontextrouters

Durch die Verknüpfung der Unicast-Varianten 2 und 4, wie dies in Abbildung 6.7 dargestellt ist, und mit Rücksicht auf die Aussagen über die örtliche Abhängigkeit der Anfragen nach kontextsensitiven Diensten in Abschnitt 6.2.4 kann auf eine Weiterleitung von Advertisements/Solicitation über Netzgrenzen hinweg verzichtet werden. Der Router versendet, wie im vorangegangenen Absatz beschrieben, Advertisements in die angebundenen Netze. Es wird davon ausgegangen, dass sich die Clients nach den Ausführungen in Abschnitt 6.2.4 in einer der zugehörigen administrativen Domänen befinden. Server, die sich außerhalb dieser administrativen Domäne befinden, können die IP-Adresse des Kontextrouters manuell beziehen. Da dadurch auf den Einsatz von Relay-Agenten bzw. Proxys verzichtet werden kann, muss auf die Belange einzelner Netzwerkprovider keine Rücksicht genommen werden. Existieren mehrere Kontextrouter im Netzwerk, kann der Client zwischen den verschiedenen Alternativen wählen. Die Kombination aus den beiden Unicast-Varianten stellt gegenwärtig das beste und effektivste Lösungskonzept dar. Diese Variante ist in allen IP-Netzwerken anwendbar. Deshalb wird vorgeschlagen, diese Lösung für das kontextsensitive Routing zu verwenden. Daraus resultierend ergeben sich für die einzelnen Komponenten der Routingarchitektur die in den folgenden Abschnitten beschriebenen Kommunikationsprozesse.

6.3.2 Kontextrouter

Der Kontextrouter stellt das zentrale Element in der Routingarchitektur dar. Er registriert die kontextsensitiven Server, beantwortet Anfragen von Clients und verwaltet die Kommunikation zwischen Client und Server. Um diese Aufgaben erfüllen zu können, muss der Kontextrouter, die in den folgenden Abschnitten beschriebenen grundlegenden Funktionalitäten erfüllen.

6.3.2.1 Advertisements

Damit ein Netzknoten die Funktionen des Kontextrouters in Anspruch nehmen kann, muss ihm dessen IP-Adresse bekannt sein. Dazu wird diese aus den in Abschnitt 6.3.1 dargelegten Gründen mit Advertisement-Paketen im Netz veröffentlicht. Eine solche Möglichkeit wird bereits bei *Mobile IP* [Perk02] angewendet. Dort wird diese Funktion über eine Erweiterung des ICMP (*Internet Control Message Protocol* [Post81b]), die so genannte *ICMP Router Discovery Message*, realisiert. Der zugehörige RFC 1256 [Deer91] beschreibt diese Erweiterung für IPv4. In Abbildung 6.8 ist ein entsprechendes ICMP-Advertisement-Paket dargestellt. Die ersten 4 Byte stimmen dabei mit denen eines herkömmlichen ICMP-Paketes überein. Im Einzelnen haben die Felder folgende Bedeutung:

Type - ist der Typ der Nachricht (Advertisement = 9).

Code - dient einer detaillierten Spezifikation. Für das „normale“ Advertisement wird per Default die 0 gesetzt.

Checksum - ist die Prüfsumme.

Num Addrs - ist die Anzahl der Routeradressen, die mit dieser Nachricht veröffentlicht werden.

Addr Entry Size - gibt die Anzahl von 32-Bit-Wörtern wieder, aus denen sich die Informationen über die jeweilige Routeradresse zusammensetzen (= Gesamtzahl der Bits aus *Router Address[i]* und *Preference Level[i]* ÷ 32).

Lifetime - ist die Dauer in Sekunden, für die die Adressen der Router als gültig betrachtet werden.

Router Address[i] - sind die gesendeten Router-IP-Adressen mit (i=1... *Num Addrs*).

Preference Level[i] - ist ein Wert für die Priorität der jeweiligen Adresse i.

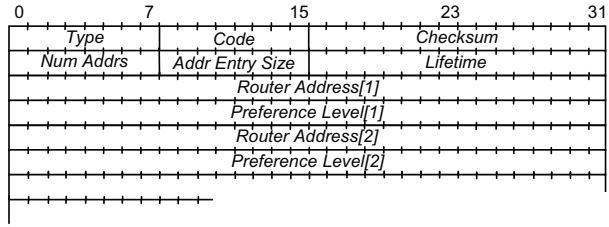


Abbildung 6.8: ICMP-Advertisement-Paket

Das Versenden eines Advertisement-Paketes reicht jedoch noch nicht aus, um den Netzknoten mitzuteilen, dass sich hinter dem Sender ein Kontextrouter verbirgt. Daraus ist lediglich ersichtlich, dass sich ein Router im Netzwerk befindet. Zur speziellen Kennzeichnung eines Kontextrouters kann beispielsweise das Feld *Code* verwendet

werden. Zu beachten ist an dieser Stelle, dass nach RFC 1256 alle Hosts solche Advertisements verwerfen. Allerdings muss auf den Clients und Servern, die kontextsensitives Routing unterstützen, spezielle Software implementiert werden, die die dafür erforderlichen Funktionalitäten ermöglicht. Dort ist dann auch das Verhalten des Hosts dahingehend zu verändern, dass Advertisement-Pakete mit einem *Code*-Feld ungleich Null nicht verworfen, sondern entsprechend ausgewertet werden. Entspricht der Wert des Feldes dem, der für einen Kontextrouter festgelegt wurde, wird die mitgelieferte IP-Adresse akzeptiert und übernommen. Somit können alle Netzknoten, die kontextsensitives Routing unterstützen, diese Advertisement-Pakete auswerten. Für alle anderen Knoten sind diese Pakete transparent und werden verworfen. Damit bleibt die Kompatibilität erhalten. Ein ähnlicher Weg ist für *Mobile IP* in [Schi03] beschrieben. Dort wird neben der Mobilitätserweiterung des Advertisementpaketes das Verhalten der Netzknoten beim Empfang eines solchen Paketes mit Hilfe verschiedener Codes (0 und 16) gesteuert. Aktuell bereits reservierte Werte für das Feld *Code* sind in [TyNo08] veröffentlicht und für die vorliegende Arbeit berücksichtigt.

Die Advertisement-Pakete werden nur in der administrativen Domäne verbreitet, in der sich der Initiator der Sendung befindet. Dazu wird bevorzugt die Multicast-Adresse 224.0.0.1 oder alternativ die *Limited*-Broadcast-Adresse verwendet. Ad-hoc-Netze unterstützen in der Regel eine Verbreitung mit diesen Adressen nicht. Dennoch können mit existierenden Multicast-Ansätzen (z. B. [RoPe99, PeBRD01]) alle Knoten eines AODV-Netzes erreicht werden. Für das Architekturkonzept müssen die Advertisement-Pakete dann entsprechend adressiert werden.

Die Verbreitung der Advertisement-Pakete erfolgt in bestimmten zeitlichen Abständen. Nach RFC 1256 liegt der maximale Abstand im Intervall zwischen 4 und 1800 Sekunden. Der Defaultwert beträgt 600 Sekunden. Natürlich besteht immer die Wahrscheinlichkeit, dass ein Knoten innerhalb dieses Zeitraums einen Kontextrouter benötigt. So ein Fall liegt zum Beispiel vor, wenn ein Rechner neu gestartet wurde. Er müsste jetzt im schlechtesten Fall die gesamte Intervallzeit auf eine Adressinformation warten. Bei Verwendung des Defaultwertes sind das 10 Minuten. Netzknoten, denen diese Zeit nicht zur Verfügung steht, können den Kontextrouter dazu veranlassen, ihnen früher diese Information zukommen zu lassen. Auf diese Funktionalität wird im nächsten Abschnitt detaillierter eingegangen.

6.3.2.2 Solicitation

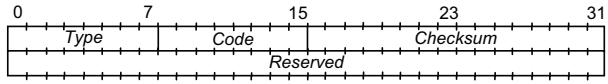


Abbildung 6.9: ICMP-Solicitation-Paket

Benötigt ein Netzknoten in kürzester Zeit die Adresse des Kontextrouters, kann er diese mit einem Solicitation anfordern. Hierbei handelt es sich um ein sehr kurzes Paket, das ebenfalls im RFC 1256 spezifiziert ist. Wie Abbildung 6.9 zeigt, besteht das Paketformat im Wesentlichen aus dem Header des ICMP. Die Bedeutung der Felder entspricht denen des oben beschriebenen Advertisement-Paketes. Der Wert von *Type* beträgt jetzt allerdings 10. Dem Header folgt ein 32 Bit langes Feld, welches reserviert ist und mit Nullen aufgefüllt wird. Im Header des unter dem ICMP liegenden IP-Paketes befindet sich als Quelladresse der Initiator der Solicitation-Nachricht.

Als Ziel wird hier die Multicast-Adresse 224.0.0.2 oder alternativ wieder die *Limited*-Broadcast-Adresse verwendet. Für Ad-hoc-Netze gelten die gleichen Aussagen wie in Abschnitt 6.3.2.1. Darüber hinaus ist die dort vorgeschlagene Modifikation des Feldes *Code* zu übernehmen.

Der Kontextrouter selbst verhält sich nach dem Erhalt eines Solicitation-Paketes wie in RFC 1256 beschrieben. Danach antwortet er mit einem Advertisement, das entweder via Unicast direkt an den Initiator des Solicitations oder mittels Multicast bzw. (alternativ) mittels Broadcast in das Netz gesendet wird.

Auch IPv6 unterstützt ICMP (RFC 4443 [CoDG06]) sowie Advertisements und Solicitations (RFC 2461 [NaNS98]), sodass die in den Abschnitten 6.3.2.1 und 6.3.2.2 beschriebenen Funktionen realisierbar sind. Die Paketformate sind entsprechend den RFCs zu berücksichtigen. Außerdem wird eine Adressierung der ICMP-Pakete mittels Broadcast nicht unterstützt.

6.3.2.3 Registrierung von kontextsensitiven Diensten

Ein Server, der kontextsensitive Dienste anbietet, soll diese auf dem Kontextrouter registrieren können. Eine Registrierung darf erst erfolgen, wenn sich der Server authentifiziert hat. Hierfür kann vorerst z. B. ein Challenge-Response-Verfahren verwendet werden, wie es beispielsweise bei GSM unter Nutzung des A3-Algorithmus zur Authentifizierung eingesetzt wird. Der Kontextrouter sendet dazu dem Server eine Zufallszahl. Der Server verschlüsselt mit dieser Zufallszahl sein Passwort und sendet das Ergebnis an den Kontextrouter zurück. Bei diesem ist das Passwort ebenfalls hinterlegt. Zum Vergleich verschlüsselt auch der Kontextrouter das Passwort mit der Zufallszahl. Stimmt das Ergebnis mit den vom Server empfangenen Daten überein, war der Authentifizierungsvorgang erfolgreich. Die anschließende Kommunikation kann bei Bedarf ebenfalls verschlüsselt durchgeführt werden. Andere Varianten der Authentifizierung und Verschlüsselung sind ebenfalls denkbar. Zu erwähnen wäre hierbei beispielsweise der *Secure Sockets Layer* (SSL), auf dem eine Reihe von Anwendungsprotokollen aufsetzen können. Derzeit existieren mehrere bewährte Lösungen, die die Aufgaben dieses Kommunikationsteils sehr gut übernehmen können. Darauf wird bei der praktischen Realisierung in Kapitel 8 zurückgegriffen.

Nach der erfolgreichen Anmeldung kann der Server die von ihm bereitgestellten kontextsensitiven Dienste und die zugehörigen Kontexttypen ablegen. Zur Übertragung der Daten wird auf herkömmliche Protokolle zurückgegriffen. Die Dienste und Kontexttypen werden dazu kodiert und können beispielsweise mit Hilfe von *Hypertext Markup Language* (HTML), XML o. ä. übertragen werden. Wichtig ist an dieser Stelle, dass eine definierte Schnittstelle in der Routerarchitektur existiert, über die diese Daten abgelegt werden können. Als einfachste Variante könnte eine binäre Kodierung für die Übertragung der Daten genutzt werden. Dabei besitzt jeder Dienst und jeder Kontexttyp eine eindeutige binäre Kennung. Unabhängig davon, wie die Kodierung letztlich realisiert wird, ist sie unbedingt in Anlehnung an die Beschreibung des Dienstes und der Kontexttypen innerhalb eines RREQ vorzunehmen. Damit können diese Daten ohne großen Umrechnungsaufwand im Kontextrouter abgelegt und miteinander verglichen werden. Das bei der Übertragung vom Server zum Client verwendete Datenformat sollte auf jeden Fall erweiterbar sein. Dadurch wird es möglich, zusätzliche in Zukunft als relevant angesehene Informationen, die der Entscheidungsfindung bei der Auswahl des Servers dienlich sein können, einzubinden.

Die übertragenen Daten werden dann in einer Tabelle auf dem Kontextrouter abgelegt. Tabelle 6.3 ist ein Vorschlag dafür, wie eine solche Tabelle aussehen könnte. Neben Dienst und zugehörigem Kontexttyp muss mindestens auch die IP-Adresse des Servers aufgelistet sein. Weitere Optionen wie Kosten, Hopanzahl (gerade beim Ad-hoc-Netz) usw. sind möglich und können in zukünftigen Erweiterungen als Entscheidungskriterium bei der Auswahl eines für den Nutzer passenden Servers dienen. Verschiedene Informationen können hierfür auch über die reguläre Routingtabelle bezogen werden (z. B. Interfaces und Hops in Tabelle 6.3). Nach der Registrierung muss sich der Server in regelmäßigen Abständen beim Kontextrouter melden und seine Einträge erneuern. Erfolgt in der vorgegebenen Zeit kein Update, wird der Eintrag wieder aus der Tabelle gelöscht. Der Startzeitpunkt des Timers ist hierbei von den anderen Servereinträgen und dem Senden von Advertisements unabhängig. Damit wird gewährleistet, dass die Wahrscheinlichkeit von Verkehrsspitzen durch gleichzeitiges Senden vieler/aller Registrierungsinformationen gering bleibt.

Server	Dienste	Kontexttyp	IP-Adresse	Interface	Hops
S1	1	4, 5, 8, 7	129.187.39.54	2	9
S2	1	1, 2, 3	141.24.92.184	1	–
	27	7			
S3	1	1, 5, 6	141.24.93.161	1	–
	13	9			
	15	5, 7, 10, 14			
S4	1	–	141.24.92.61	1	–
S5	1	2, 3, 9, 10	141.30.2.2	3	10
	15	5, 7			
	29	12, 28, 36			
S6	1	1, 4	130.149.4.134	2	8

Tabelle 6.3: Beispiel für eine Server-Routing-Tabelle

6.3.2.4 Anfrage nach kontextsensitiven Diensten

Die Anfrage nach einem kontextsensitiven Dienst erfolgt mit einem erweiterten AODV-RREQ vom Client aus. Erhält der Kontextrouter ein solches Paket, liest er die darin enthaltenen Informationen (Dienst, Kontexttypen) aus. Um die Auswahl eines geeigneten Servers differenzierter und flexibler gestalten zu können, wird die Übertragung einer weiteren Information vorgeschlagen. Dabei handelt es sich um die Priorisierung der einzelnen Kontexttypen. Die Notwendigkeit einer Priorisierung ergibt sich beispielsweise aus deren in Abschnitt 4.4 beschriebenen unterschiedlichen Bedeutung für die Anwendung. Danach ist z. B. nicht jeder vom Client bzw. Nutzer bereitgestellter Kontexttyp für das Erbringen des geforderten Dienstes nötig. Darüber hinaus können in einem Netzwerk zur selben Zeit auch mehrere gleichartige Dienste angeboten werden, welche aber unterschiedliche Kontexttypen berücksichtigen (siehe auch den folgenden Abschnitt 6.3.2.5). In diesem Fall unterstützt eine Priorisierung die Auswahl eines geeigneten Servers. Zu beachten ist allerdings, dass die in Abschnitt 4.5 erläuterten obligatorischen Kontexttypen zur Erbringung des Dienstes vom Client bereitgestellt und bei der Anfrage zwingend mitgesendet werden. Da Anwendungen in der Regel auf die jeweiligen Dienste abgestimmt sind, ist

davon auszugehen, dass ihnen diese Kontexttypen bekannt sind. Im Folgenden wird deshalb vorausgesetzt, dass die obligatorischen Kontexttypen per Default in einer Dienstanfrage enthalten sind.

Im Anschluss an die Anfrage muss verglichen werden, inwieweit der angeforderte Dienst mit den beim Router registrierten Diensten übereinstimmen. Über eine entsprechende Auswahllogik wird dann entschieden, welcher Server den geeignetsten Dienst anbietet. Hierfür legt der Kontextrouter eine interne Reihenfolge, eine „Serverrangliste“, für in Frage kommende Server fest. Aus Gründen der Flexibilität sollte die Implementierung der Auswahllogik auf dem Router so gestaltet werden, dass sie auf bestimmte Anforderungen hin angepasst werden kann. Auf die Auswahllogik wird in Abschnitt 6.3.2.5 detaillierter eingegangen.

Ist eine Auswahl getroffen worden, sendet der Router ein erweitertes AODV-RREP an den Client zurück, welches entweder den passendsten Server mit den dort unterstützten Kontexttypen enthält oder die Anfrage negativ beantwortet. Eine Antwort ist dann negativ, wenn kein entsprechender Dienst auf dem Kontextrouter registriert ist (siehe auch Abschnitt 6.3.4). Eine wiederholte Anfrage durch den Client ist dann möglich, wenn der Nutzer den vorgeschlagenen Server nicht in Anspruch nehmen will. Daraufhin bietet der Kontextrouter dem Client als Alternative den Server an, der im Auswahlprozess den nächsten Rang belegt hat. Handelt es sich um den letzten alternativen Server, wird in der Antwort das so genannte *No Alternatives*-Flag gesetzt, welches mit dieser Routingarchitektur neu eingeführt wird. Ein Vorschlag für den konkreten Aufbau der erweiterten RREQ-/RREP-Pakete erfolgt in Abschnitt 6.3.4. Dort wird auch noch einmal detailliert auf den Anfrageprozess aus Sicht des Clients eingegangen. Mit dem Antwortpaket übergibt der Router auch einen Identifikator und eine Sitzungsnummer an den Client. Anhand dieser Werte ist dem Kontextrouter eine Zuordnung zwischen Client und Server möglich. Eine detaillierte Erklärung der Funktionsweise erfolgt in Abschnitt 6.3.2.6.

Nachdem der Anfrage-/Antwortprozess beendet wurde, kann die Serverrangliste wieder gelöscht werden, sofern diese nicht für zusätzliche optionale Funktionen des Kontextrouters benötigt wird. Aus Gründen der Aktualität muss aber die Liste eine begrenzte „Lebensdauer“ besitzen. Erhält der Router ein RREQ ohne Kontexterweiterung, hat er diese Anfrage wie ein herkömmliches Routingpaket zu behandeln. D. h., es wird auf das Paket mit einem RREP geantwortet, sofern die Route zum Ziel bekannt ist. Ist dies nicht der Fall, wird das RREQ-Paket mit einem Broad-/Multicast an die benachbarten Knoten weitergeleitet.

6.3.2.5 Serverauswahl

Die Auswahl eines kontextsensitiven Servers, der einer Anforderung eines Clients entspricht, kann auf unterschiedliche Weise erfolgen. So spielen neben dem vom jeweiligen Server unterstützten Dienst und den eigentlichen Kontexttypen auch deren vom Nutzer vergebenen Prioritäten eine Rolle. Der Einfluss der Prioritäten, die den Kontexttypen zugeordnet wurden, und verschiedene optionale Werte aus der Routingtabelle können während des Auswahlprozesses unterschiedlich berücksichtigt werden. So sind Kriterien wie Anzahl der Hops zum Server, Kosten und QoS-Unterstützung als zusätzliche Entscheidungsmerkmale möglich. Diese können auch miteinander kombiniert werden. Die Liste der Kriterien ist damit beliebig erweiterbar. Die Auswahlkriterien sind so vielfältig, dass eine Implementierung der

Auswahllogik sehr flexibel gestaltet sein sollte. Dem Provider muss es damit ermöglicht werden, diese entsprechend den Bedürfnissen des Nutzers anzupassen. Dazu können auf dem Kontextrouter beispielsweise verschiedene alternative Auswahlalgorithmen angeboten werden. Es sind auch Implementierungen denkbar, mit denen die kontextsensitive Anwendung bzw. der Nutzer über die gewünschte Auswahllogik entscheiden und diese auswählen kann. Eine weitere Herausforderung ist damit die Entwicklung allgemeingültiger Algorithmen, die eine zukünftige Erweiterung auf zusätzliche Auswahlkriterien auf einfache Weise ermöglichen.

Das eigentliche Ziel des Auswahlalgorithmus besteht darin, nach einer Anfrage durch einen Client mittels erweitertem RREQ-Paket eine Rangfolge der in Frage kommenden Server zu erstellen. Diese Rangfolge soll dabei mit der Eignung der dort bereitgestellten Dienste für den Nutzer korrelieren. Der angeforderte Dienst gilt hierbei als Ausschlusskriterium. Dem Nutzer braucht kein Dienst angeboten zu werden, den er nicht angefordert hat. Daneben können die Kontexttypen und deren Prioritäten direkt zu einer Bewertung der Eignung eines Dienstes und damit als Kriterien zum Erstellen der Rangliste beitragen. Die Einteilung der Prioritäten erfolgt in 8 Stufen. Dazu werden, wie später in Abschnitt 6.3.4 beschrieben, 3 Bit bereitgestellt. Eine feinere Granularität ist für den Nutzer wenig sinnvoll. Schließlich muss er über die Priorisierung entscheiden. Alternativ kann dies aber auch von der jeweiligen Anwendung übernommen werden. Im Folgenden soll anhand von drei Beispielen verdeutlicht werden, wie die Berücksichtigung der Kontexttypen und ihrer Prioritäten zu unterschiedlichen Ranglisten führen kann. Kontexttypen mit der Priorität 0 bilden dabei eine Besonderheit. Damit sind solche Kontexttypen gemeint, die nicht von der Anwendung bzw. dem Nutzer gefordert sind, aber zur Verfügung stehen. Solche Informationen können dann beispielsweise vom Server bzw. Provider für andere Dienste (z. B. Kontextinformationen über die Interessen des Nutzers für eine angepasste Werbung) gezielt genutzt werden.

Lineare Auswahl

Die einfachste Form der Auswahl berücksichtigt als Entscheidungskriterium die absolute Größe der Priorität einzelner Kontexttypen. Die Rangfolge der Server ergibt sich nach dem in Abbildung 6.10 dargestellten Schema. Auf eine mathematische Darstellung des Auswahlverfahrens wird an dieser Stelle verzichtet, da es sich hier um einen einfachen Größenvergleich zwischen verschiedenen Werten handelt. Der Algorithmus arbeitet wie folgt:

1. Zuerst werden die vom Client bereitgestellten Kontexttypen (KtC) mit denen der Server, die den gesuchten Dienst anbieten, verglichen. Dazu wird für jeden betreffenden Server eine Reihenfolge der geforderten und unterstützten Kontexttypen (KtS) entsprechend ihrer Priorität (P) gebildet. Diese beginnt der höchsten und endet mit der kleinsten Priorität.

2. Server, deren unterstützte Kontexttypen mit den gleichen Prioritäten beginnen, bilden eine gemeinsame Gruppe. Von den so entstandenen Gruppen wird wieder eine Rangfolge erstellt. Die Gruppe mit der höchsten Priorität steht an oberster Stelle in der Rangliste. Danach kommt die Gruppe mit den Servern, deren Kontexttypen die nächst kleinere Priorität besitzen usw.

3. Jetzt erfolgt die gleiche Vorgehensweise für jede einzelne Gruppe, sofern dort mehr als ein Server enthalten ist. Dabei müssen jedoch die bereits verglichenen Prioritäten unberücksichtigt bleiben. Innerhalb jeder Gruppe wird die Priorität gemäß Punkt 2 bestimmt. Damit ergibt sich eine Rangfolge innerhalb einer jeden Gruppe. Zu beachten ist hierbei, dass Server bzw. deren Dienste, die keine weiteren Kontettypen unterstützen, den Wert Null erhalten und somit den Platz am Ende der Rangliste innerhalb der aktuell betrachteten Gruppe einnehmen. Die Vorgehensweise wird dann so oft wiederholt, bis nur einzelne oder vollkommen gleichwertige Server übrig bleiben.

4. Am Ende sind alle Server in eine Rangfolge eingeordnet. Tritt der Fall auf, dass zwei gleichwertige Server eine gemeinsamen Rang einnehmen, können zusätzliche Entscheidungsmerkmale (z. B. Hopanzahl zum Server, Lastverteilung, ...) verwendet werden, um eine optimale Serverauswahl treffen zu können.

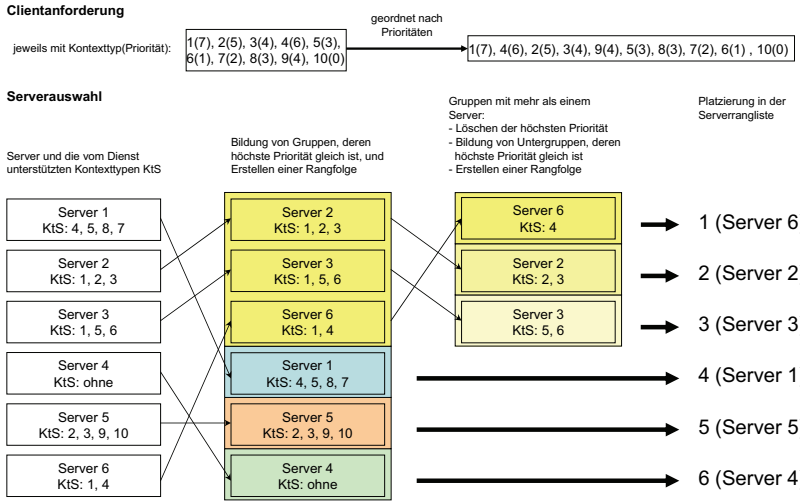


Abbildung 6.10: Entscheidungsalgorithmus für die lineare Auswahl

Nachteilig bei dem oben beschriebenen Auswahlverfahren ist, dass nur die Wertigkeiten der Prioritäten nicht aber die Menge der unterstützten Kontexttypen berücksichtigt werden. Somit wird beispielsweise ein Server bevorzugt, der vielleicht nur einen einzigen Kontexttyp, welcher aber die höchste Priorität besitzt, unterstützt. Ein Server, der eine Vielzahl von geforderten Kontexttypen bereitstellt, denen aber eine kleinere Priorität zugeordnet ist, wird erst dahinter in der Rangliste aufgenommen. Ein Verfahren, das die Anzahl der unterstützten Kontexttypen mit berücksichtigt, wird im folgenden Abschnitt beschrieben.

Kumulierte Auswahl

Bei diesem Auswahlalgorithmus wird versucht, neben den Prioritäten auch die Anzahl der bei den einzelnen Servern zur Verfügung stehenden Kontexttypen zu berücksichtigen. Mit anderen Worten – es wird ein Maßstab zum Ausdruck der mittleren Kontextdichte gesucht. Dazu kann zum Beispiel ein prozentualer Vergleich durchgeführt werden. Ein Server, der einen Dienst anbietet, dessen unterstützte Kontexttypen und deren Prioritäten genau den geforderten des Clients entsprechen, erhält damit 100%.

Für die Menge der vom Client geforderten n Kontexttypen kt gilt hierbei:

$$KtC = \{kt_i\} \qquad \text{mit} \quad i = 1, \ldots, n; \quad i \in \mathbb{N} \tag{6.1}$$

Jedem einzelnen kt_i ist die Priorität $P(kt_i)$ zugeordnet. Vorausgesetzt ein Server k ($k \in \mathbb{N}$) bietet den gesuchten Dienst des Clients an, dann gilt für die Anzahl m der von ihm unterstützten Kontexttypen kt:

$$KtS_k = \{kt_i\} \qquad \text{mit} \quad i = 1, \ldots, m; \quad i \in \mathbb{N} \tag{6.2}$$

Ein Vergleich zwischen bereitgestellten Kontexttypen eines Clients und den dazu vorhandenen Angebot eines Servers k führt dann allgemein zu:

$$Match_k(kt) = \begin{cases} 1, \text{ wenn} & kt \in KtS_k \\ 0, \text{ wenn} & kt \notin KtS_k \end{cases} \tag{6.3}$$

Damit ergibt sich für den Server k folgender Vergleichswert für die Auswahlliste:

$$X_k = \frac{\sum\limits_{\forall kt_i \in KtC} (P(kt_i) + 1) \cdot Match_k(kt_i)}{\sum\limits_{\forall kt_i \in KtC} (P(kt_i) + 1)} \cdot 100\% \tag{6.4}$$

X_k ist hierbei ein prozentualer Wert, der ausdrückt, inwieweit der Dienst des Servers k mit der Anforderung des Clients bezüglich der Kontexttypen übereinstimmt. Der Zähler ergibt sich aus der Summe der Prioritäten der vom Client bereitgestellten Kontexttypen, die von dem angebotenen Dienst auf dem Server k berücksichtigt werden. Der Nenner setzt sich aus der Summe aller Prioritäten der vom Client unterstützten Kontexttypen zusammen. Die einzelnen Prioritäten werden um jeweils 1 erhöht, damit der Nenner stets größer als Null ist. Mit Gleichung 6.4 ergibt sich die in Tabelle 6.4 dargestellte Serverrangliste. Die Werte für X_k sind hierbei auf die erste Dezimalstelle nach dem Komma gerundet. Verbleiben im Ergebnis einmal gleichrangige Server, wird wie in Abschnitt 6.3.2.5 unter Punkt 4 des Auswahlalgorithmus verfahren.

Mit den Ergebnissen in Tabelle 6.4 werden jetzt zwar alle Kontexttypen berücksichtigt, allerdings ist hier keine Möglichkeit gegeben, die einzelnen Prioritäten unterschiedlich zu wichten. Eine Lösung dafür wird im nächsten Abschnitt dargestellt.

k	$KtS_k \cap KtC$ (mit Angabe der Priorität)	X_k in %	Platzierung
1	4(6), 5(3), 8(3), 7(2)	40,0	2
2	1(7), 2(5), 3(4)	42,2	1
3	1(7), 5(3), 6(1)	31,1	5
4	–	0,0	6
5	2(5), 3(4), 9(4), 10(0)	37,7	3
6	1(7), 4(6)	33,3	4

Tabelle 6.4: Serverrangliste nach kumulierter Auswahl

Gewichtete Auswahl

Um höheren Prioritäten auch bei der Auswahlentscheidung eine größere Wichtung zukommen lassen zu können, muss die Gleichung 6.4 modifiziert werden. Dazu eignet sich beispielsweise das Potenzieren der Prioritäten. Je größer die Wahl der Exponenten ist, desto höher wird der Einfluss großer Prioritäten. Natürlich können für die Prioritäten einzelner Kontexttypen auch unterschiedliche Exponenten verwendet werden. Die folgende Gleichung zeigt ein Beispiel für die Nutzung eines einheitlichen Exponenten. Vorausgesetzt werden hier wieder die Gleichungen 6.1, 6.2 und 6.3:

$$X_k = \frac{\sum\limits_{\forall kt_i \in KtC} [(P(kt_i)+1) \cdot Match_k(kt_i)]^j}{\sum\limits_{\forall kt_i \in KtC} [P(kt_i)+1]^j} \cdot 100\% \qquad \text{mit} \quad j > 1 \tag{6.5}$$

Der prozentuale Wert X_k stellt wiederum einen Vergleichswert für den kontextsensitiven Dienst eines Servers k in Bezug auf die Übereinstimmung zu dem vom Client geforderten Dienst und die hierfür bereitgestellten Kontexttypen dar. Allerdings wird jetzt im Gegensatz zu Gleichung 6.4 die unterschiedliche Wichtung der Prioritäten berücksichtigt. Dazu werden die Prioritäten der einzelnen Kontexttypen aus dem bereits erläuterten Grund inkrementiert, mit j potenziert und anschließend summiert. Der Wert j stellt dabei das Maß für die Wichtung der Prioritäten dar. Je nach Anforderung kann dieser Parameter durch den Nutzer selbst bestimmt oder aber mit dem implementierten Algorithmus festgelegt werden. Bei gleichwertigen Servern, d. h. die Werte X_k sind gleich, wird wiederum wie in Abschnitt 6.3.2.5 unter Punkt 4 der Auswahllogik verfahren.

Tabelle 6.5 zeigt, welches Ergebnis sich nun für das obige Beispiel ergeben würde, wenn $j = 2$ gesetzt wird. Auch hier ist wieder X_k auf die erste Dezimalstelle nach dem Komma gerundet. Im Ergebnis der Serverauswahl ist zwar auch hier der erste Platz durch Server 2 belegt, allerdings belegt nun Server 6 den zweiten Rang, da dies der einzige Server ist, dessen Dienst die zwei Kontexttypen mit den beiden höchsten Prioritäten unterstützt. Obwohl andere Server mehr Kontexttypen berücksichtigen, erfolgt durch die geringere Wichtung der Prioritäten eine Abwertung.

Bei allen drei Algorithmen werden Anfragen ohne die Angabe eines Kontexttypes nicht berücksichtigt. Es würde sich hierbei um eine allgemeine Dienstanfrage handeln. Der Router kann dieser Anfrage einen beliebigen Server zuordnen, der diesen Dienst erbringt. Eine Auswahl bei mehreren alternativen Servern könnte dabei

k	$KtS_k \cap KtC$ (mit Angabe der Priorität)	X_k in %	Platzierung
1	4(6), 5(3), 8(3), 7(2)	36,7	3
2	1(7), 2(5), 3(4)	51,0	1
3	1(7), 5(3), 6(1)	34,3	5
4	–	0,0	6
5	2(5), 3(4), 9(4), 10(0)	35,5	4
6	1(7), 4(6)	46,1	2

Tabelle 6.5: Serverrangliste nach gewichteter Auswahl

mit Rücksicht auf die Lastverteilung erfolgen. Sicherlich gibt es noch eine Vielzahl von Auswahlmöglichkeiten. Hinzu kommt, dass die Algorithmen miteinander kombiniert werden können. Bisher wurden zur Entscheidungsfindung auch noch keine Routinginformationen einbezogen. Für zukünftige Entwicklungen bietet dies einen interessanten Ansatzpunkt. Letztlich muss die Praxis zeigen, welches Verfahren am geeignetsten ist.

Nachdem die Auswahl des Servers abgeschlossen ist, wird dies dem Client mitgeteilt. Wie die daran anschließende Kommunikation mit dem Server erfolgen kann, beschreibt der folgende Abschnitt.

6.3.2.6 Weiterleitungsfunktion

Hat der Client die Antwort des Kontextrouters empfangen und den ausgewählten Server akzeptiert, kann er die Kommunikation mit ihm initiieren. Wie in den Abschnitten 5.3 und 6.1 beschrieben, wären dafür sowohl eine direkte als auch eine indirekte Kommunikation denkbar. Im zuletzt genannten Abschnitt wurde die indirekte Kommunikation als vorteilhafter analysiert. Deshalb wird sich im Folgenden auf diese bezogen.

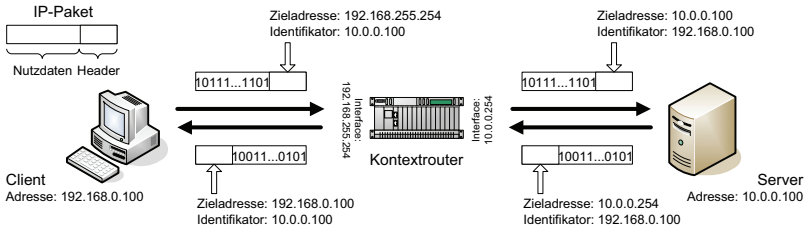


Abbildung 6.11: Weiterleitung durch den Kontextrouter

Der Client sendet seine Daten weiterhin an den Kontextrouter und dieser leitet sie an den entsprechenden Server weiter. Abbildung 6.11 beschreibt dieses Szenario. Empfängt der Kontextrouter ein IP-Paket, ordnet er dies anhand der Quelladresse und des mit dem Client für diese Kommunikation vereinbarten Identifikators dem ausgewählten Zielserver zu. Im aktuellen Konzept handelt es sich bei diesem Identifikator um die IP-Adresse des Zielservers. Der Kontextrouter tauscht nun die Adresse des IP-Paketes aus. Dieses erhält als Ziel die IP-Adresse des Servers. Der Identifikator

wird dabei ebenfalls ersetzt. Für diesen wird die IP-Adresse des Clients verwendet. In entgegengesetzter Richtung werden die vom Server zurückgesendeten Pakete wieder mit Hilfe des Identifikators eindeutig dem entsprechenden Client zugeordnet. Der Kontextrouter kann dadurch das IP-Paket direkt an den Client weiterleiten. An diesem Beispiel lassen sich folgende Eigenschaften für den Identifikator ableiten:

- Der Identifikator ist ein Wert, der vom Kontextrouter festgelegt wird und einem IP-Interface zugeordnet ist.
- Ein Client kann gleichzeitig mit verschiedenen Servern kommunizieren. Der Kontextrouter unterscheidet dies anhand der Identifikatoren. Diese korrelieren bei der Kommunikation zwischen Client und Kontextrouter mit den jeweiligen Servern.
- Ein Server kann gleichzeitig mit verschiedenen Clients kommunizieren. Allerdings korrelieren jetzt die Identifikatoren bei der Kommunikation zwischen Kontextrouter und Server mit dem jeweiligen Client.
- Eine gleichzeitige Kommunikation zwischen Anwendungen auf einem Client und verschiedenen Diensten auf ein und demselben Server ist möglich. Eine Unterscheidung erfolgt dann genauso wie in herkömmlichen IP-Netzen in höheren Protokollschichten (z. B. mit Hilfe des Ports).

Der Kontextrouter kann somit für die Kommunikation mit verschiedenen IP-Interfaces von Clients und Servern die gleichen Identifikatoren verwenden. Diese müssen aber in der Kombination mit der jeweiligen Adresse des Interfaces eineindeutig sein. Außerdem ist es nicht zwingend notwendig IP-Adressen zur Kennzeichnung der Identifikatoren zu verwenden. Stattdessen können auch beliebige Werte eingesetzt werden, sofern diese die aufgeführten Anforderungen erfüllen.

Tritt während der Kommunikation mit dem Server der Fall ein, dass der Kontextrouter keine Antwort innerhalb einer bestimmten Zeit erhält, sendet er eine Meldung an den Client (z. B. mit einer entsprechenden ICMP-Nachricht) zurück und löscht alle zu diesem Server gehörenden Einträge in seinen Tabellen. In diesem Fall ist davon auszugehen, dass der Server vom Netz getrennt ist. Alternativ kann der Kontextrouter die folgende Kommunikation auch auf einen gleichwertigen Server, d. h. der den gleichen Dienst mit denselben Kontexttypen unterstützt, umleiten bzw. rerouten. Wie in [Renh06] richtig festgestellt wurde, ist an dieser Stelle problematisch, dass die bis dahin empfangenen Informationen und aufgenommenen Kontextdaten verloren gehen. Bei einigen kontextsensitiven Diensten könnte dies zu Problemen führen. Hat der Nutzer beispielsweise einen Navigationsdienst verwendet, kennt der alternative Server zwar die aktuelle Position des Nutzers, die gewählte Route ist ihm aber unbekannt. Da es von der Anwendung abhängig ist, ob die „historischen“ Daten benötigt werden, sollte die Entscheidung für ein Rerouting dem Client überlassen werden. So kann dieser schon mit dem Senden des erweiterten RREQ durch Setzen eines Flagbits angeben, ob bei Ausfall eines Servers automatisch auf einen alternativen Server umgeleitet werden soll. Dieses Flag wird in den folgenden Abschnitten als *No Reroute*-Flag (NR-Flag) bezeichnet. Ist es gesetzt, wird kein automatisches Rerouting auf einen alternativen Server gewünscht. Aus Abbildung 6.2 ist bekannt, dass im Paketkopf der RREQ- und RREP-Pakete reservierte Bereiche existieren. Davon

kann ein Bit für das NR-Flag verwendet werden. Es reicht, dieses Flag innerhalb der RREQ-Pakete vorzusehen. Für die Realisierung des Konzeptes wird vorgeschlagen, das erste freie Bit nach den regulären Flags im Paketkopf zu verwenden.

Erhält der Kontextrouter ein (ungültiges) IP-Paket, welches zwar an ihn adressiert ist, wofür er aber keine Zuordnung zwischen Absender-IP-Adresse und Identifikator besitzt, muss dieses Paket verworfen werden. Das gilt sowohl für Pakete, die vom Client, als auch für solche, die vom Server gesendet wurden. Eine Reaktion des Kontextrouters beispielsweise über ICMP ist nicht notwendig. IP-Pakete, deren Optionen keinen Identifikator enthalten, werden als reguläre Daten wie bei einem herkömmlichen Router weitergeleitet.

Für ein Rerouting der IP-Pakete bei Ausfall des Zielservers reicht der Identifikator jedoch nicht aus. Infolge der Umleitung zu einem alternativen Server, kann der Fall auftreten, dass dieser Server bereits mit dem Client kommuniziert. Ausgangspunkt für ein Beispiel ist hier wieder Abbildung 6.11. Es wird angenommen, dass der dort dargestellte Server (IP-Adresse: 10.0.0.100) ausgefallen ist und die Daten nun alternativ an den in Abbildung 6.12 dargestellten Server (IP-Adresse: 10.0.0.200) weitergeleitet werden. Dieser Server kommuniziert aber bereits mit dem Client. Der Kontextrouter kann in diesem Fall nicht mehr eindeutig entscheiden, welche Pakete am Interface 10.0.0.254 zu der bereits bestehenden und welche zu der neuen Kommunikation gehören. Die IP-Adressen und Identifikatoren sind bei allen Paketen gleich. Um dem Kontextrouter die Möglichkeit zu geben, trotzdem die entsprechenden IP-Pakete der ursprünglichen Kommunikation zuzuordnen, muss folglich eine weitere Kennung zu deren Identifikation übertragen werden. Diese wird als Sitzungsnummer bezeichnet und vom Kontextrouter vergeben, wenn dieser auf eine Anfrage des Clients antwortet. Damit wird die Kommunikation über das gesamte Netzwerk hinweg eineindeutig. Die Kommunikation zwischen Kontextrouter und Client bleibt trotz der Umleitung unverändert, d. h. die IP-Adresse und der Identifikator sind hier gleich geblieben. Dies ist beabsichtigt, da der Client von einem möglichen Serverwechsel nichts bemerken soll.

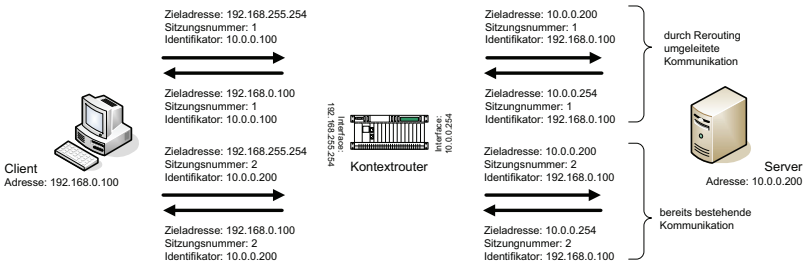


Abbildung 6.12: Paketumleitung bei Serverausfall

In Abbildung 6.12 sind die umgeleiteten Pakete mit der Sitzungsnummer 1 und die Pakete der regulären Kommunikation mit 2 gekennzeichnet. Natürlich könnte an dieser Stelle von Anfang an mit eineindeutigen Nummern gearbeitet und auf den Identifikator verzichtet werden. Nachteilig wäre in diesem Fall jedoch, dass der Kontextrouter alle Kommunikationsbeziehungen für die Dauer ihres Bestehens speichern müsste, damit die IP-Pakete den Kommunikationsbeziehungen zugeordnet werden

können. Dagegen ist bei Verwendung eines Identifikators eine Zuordnung der IP-Pakete für Übertragungen, die kein Rerouting benötigen, möglich, ohne dass dazu Informationen auf dem Router vorgehalten werden müssen. Lediglich bei einer Umleitung wird die Sitzungsnummer benötigt. Diese Nummern müssen für jede Kommunikationsbeziehung eines Client eindeutig sein. Verschiedene Clients können auch gleiche Nummern verwenden.

Für die Übertragung des Identifikators wird sowohl auf der Teilstrecke Client–Router als auch auf der Teilstrecke Router–Server das beim Internetprotokoll vorgesehene Feld für Optionen genutzt. Beim IPv6 können solche Optionen durch die Nutzung eines *Extended Headers* angegeben werden. Dort sind dann jeweils die Vorgaben aus den entsprechenden Spezifikationen zu berücksichtigen. Wenn die IP-Adressen, wie hier vorgeschlagen, als Identifikatoren verwendet werden, erübrigt sich eine Kontrolle für eine doppelte Vergabe. Die Zuordnungen sind dann für diese Weiterleitung eindeutig. In Abbildung 6.13 ist der Aufbau des *Options*-Feldes für das IPv4 dargestellt. Verpflichtend ist hierbei das erste Byte für den *Option-Type*. Dieses ist wiederum in drei Felder gegliedert und folgt direkt der Ziel-Adresse (*Destination Address*) im IP-Paketheader. Die einzelnen Felder haben dabei folgende Bedeutung:

Copied Flag (C) - wird gesetzt, wenn die Option bei einer Fragmentierung in jedes Teilfragment kopiert werden soll.

Option Class (OC) - Folgende Klassen sind vergeben: 0 = *Control* (Steuerung); 2 = *Debbuging and Measurement* (Fehlersuche und Messen); 1 und 3 sind reserviert für zukünftige Anwendungen.

Option Number (ON) - Die Werte 0–4 und 7–9 sind bereits für folgende Optionen vergeben: *End of Operation, No Operation, Security, Loose Source Routing, Strict Source Routing, Record Route Stream ID* und *Internet Timestamp*.

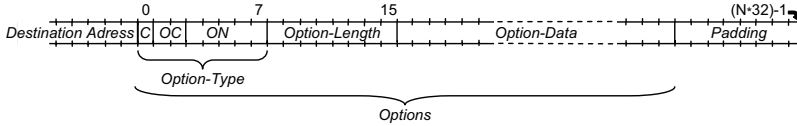


Abbildung 6.13: Das *Options*-Feld im IPv4

Ein weiteres Byte ist nötig, sobald zusätzlich auch Bereiche zur Übertragung weiterer Optionsdaten (*Option-Data*) benötigt werden. Das zugehörige Feld heißt *Option-Length* und gibt die Gesamtzahl aller für die Optionen verwendeten Bytes an. Das *Padding*-Feld ist dann so aufzufüllen, dass die Länge des gesamten IP-Paketkopfes ein Vielfaches von 32 Bit ergibt.

Abbildung 6.14 zeigt einen Vorschlag für die Implementierung des Optionen-Feldes innerhalb der Architektur für das kontextsensitive Routing. Das *Copied Flag* ist gesetzt, damit die Optionen eines jeden Teils eines fragmentierten IP-Pakets den Kontextrouter erreichen und von ihm ausgewertet werden können. Das Feld für *Option Class* kann auf 0 gesetzt werden, da es sich beim kontextsensitiven Routing um eine Steuerungsfunktion handelt. Die *Option Number* kann bis auf die vergebenen Werte (0-4 und 7-9) frei gewählt werden. Mit Rücksicht auf den RFC 4727

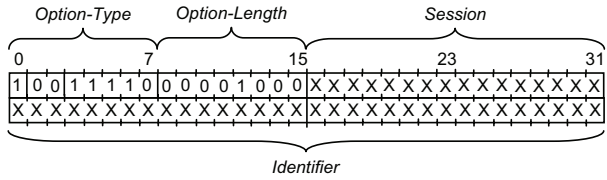


Abbildung 6.14: Belegung des *Options*-Feldes (IPv4)

[Fenn06] und die Vorgaben der IANA wurde 30 gewählt und dem *Context Sensitive Routing* zugeordnet. Diese *Option Number* steht speziell für Testzwecke zur Verfügung und muss im Falle eines zukünftigen produktiven Einsatzes geändert werden. Das Längenfeld hat den Wert 8, da davon ausgegangen wird, dass 2 Byte für die Sitzungsnummer (*Session*) und 4 Byte für die Vergabe von Identifikatoren (*Identifier*) ausreichen. Hinzu kommt, dass sowohl das Byte für das *Option-Types*- als auch für das *Option-Length*-Feld mitgezählt werden. Das *Padding*-Feld des IP-Paketes entfällt. Für IPv6 muss die Formatierung entsprechend angepasst werden, was zum einen die Länge des Identifikators (128 Bit) und zum anderen die Einbindung in den *Extension Header* betrifft.

Für die Kommunikation zwischen Kontextrouter und kontextsensitivem Server könnte die eindeutige Zuordnung der Daten auch mit Hilfe des *Network Address Translation* (NAT) realisiert werden. Allerdings würde hierbei auf Protokolle höherer Schichten wie TCP/UDP, die sich in der Transportschicht befinden, zurückgegriffen. Dies erhöht die Rechenlast und Komplexität des Routers, weil damit zwischen Server und Client keine direkte Ende-zu-Ende-Kommunikation auf der Transportschicht möglich ist. Mit der hier vorgestellten Lösung und der Verwendung von Identifikatoren auf der Vermittlungsschicht werden auch die durch NAT verursachten Probleme (siehe RFC 3027 [SrHo01]) vermieden. Das kontextsensitive Routing bleibt also lediglich auf die Netzfunktionen beschränkt. Für die höheren Protokollschichten, die Ende-zu-Ende-Funktionen erfüllen, ist es vollkommen transparent.

Abbildung 6.15 zeigt ein *Message Sequence Chart* (MSC), das die einzelnen Kommunikationsprozesse des Architekturkonzeptes anhand eines einfachen Beispiels mit einem Server, einem Client und einem Kontextrouter zusammenfassend beschreibt. Hier ist der Fall dargestellt, dass Client und kontextsensitiver Server die IP-Adresse des Kontextrouters über Advertisements erhalten. Danach authentifiziert sich der Server entsprechend dem Vorschlag aus Abschnitt 6.3.2.3 mittels Challenge-Response-Verfahren. Im Anschluss an die erfolgreiche Authentifizierung registriert der Server seine Dienste und die von ihnen unterstützten Kontexttypen beim Router. Zu einem beliebigen späteren Zeitpunkt fragt der Client beim Kontextrouter mit Hilfe eines erweiterten AODV-RREQ nach einem kontextsensitiven Dienst an und erhält über ein erweitertes AODV-RREP eine positive Antwort. Der darin vorgeschlagene Server wird vom Client akzeptiert und es erfolgt nun über den Kontextrouter der Zugriff auf den angeforderten Dienst, d. h. Client und Server können miteinander kommunizieren.

Wie bereits erwähnt, wäre prinzipiell mit dem hier vorgeschlagenen Konzept auch eine direkte Kommunikation zwischen Client und Server unabhängig vom Kontextrouter möglich. Dabei wird auf die Weiterleitungsfunktion eines Routers verzichtet.

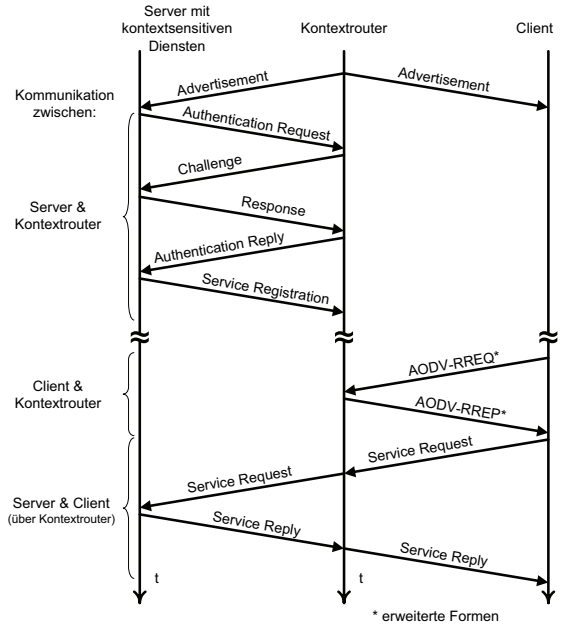


Abbildung 6.15: Kommunikationsprozesse des Architekturkonzeptes

Es sind dann auch keine Sitzungsnummer und kein Identifikator im RREP-Paket mehr nötig. Der Client teilt dafür im RREP zusätzlich die Zieladresse des ausgewählten Servers mit. Damit können dann Client und Server direkt miteinander kommunizieren – allerdings mit den in Abschnitt 6.1 aufgezählten Nachteilen.

Das hier vorgestellte Modell des Kontextrouters kann natürlich noch erweitert werden. So ist beispielsweise vorstellbar, dass auch eine Liste alternativer Kontextrouter vorgehalten wird. Diese Router können wiederum ihr Wissen über kontextsensitive Server austauschen. Damit wäre es dann auch möglich, Anfragen, die von einem Kontextrouter nicht beantwortet werden können, an einen anderen Kontextrouter weiterzuleiten. Daneben sind auch Strategien zur Last- und Funktionsteilung denkbar. Die Liste könnte noch beliebig fortgeführt werden, soll aber Thema zukünftiger Forschungsarbeiten sein.

6.3.3 Kontextsensitiver Server

Nachdem die Funktionen des Kontextrouters festgelegt wurden, definiert dieser Abschnitt die Mindestanforderungen an einen kontextsensitiven Server, damit dieser in die kontextsensitive Routingarchitektur eingebunden werden kann. Der Server muss die von einem Kontextrouter empfangenen Advertisement-Pakete auswerten können. Die so erhaltenen IP-Adressen werden abgespeichert. Sie bleiben für eine bestimmte Zeit gültig. Nach RFC 1256 wären das beispielsweise 30 Minuten. Erhält der Server innerhalb dieser Zeit keine weiteren Advertisements von einem Router, ist dieser aus der Liste der bekannten Kontextrouter wieder zu entfernen. Besitzt der Server keine

Adresseinträge (z. B. nach einem Neustart), kann er durch Senden von Solicitation-Paketen eventuell im Netz vorhandene Kontextrouter dazu veranlassen, ihm ihre IP-Adresse mitzuteilen. Zu beachten ist, dass Solicitations auch wiederholt gesendet werden dürfen, sofern innerhalb einer bestimmten Zeit keine Reaktion in Form einer Antwort durch einen Kontextrouter erfolgte. Allerdings ist hierfür die Anzahl der Solicitation-Pakete begrenzt. Im Wesentlichen sollte sich bei der Umsetzung des Verhaltens beim Empfang von Advertisement-Paketen und der zu realisierenden Solicitations an den RFC 1256 gehalten werden. Zusätzlich ist vorzusehen, dass dem Server die Adressen von kontextsensitiven Routern auch manuell übergeben werden können.

Der Server kann nun auf allen ihn bekannten Kontextroutern seine kontextsensitiven Dienste und die von ihnen unterstützten Kontexttypen registrieren, sofern er dort die entsprechenden Zugangsrechte besitzt. Zur Registrierung muss er sich beim Kontextrouter authentifizieren. Wie bereits in Abschnitt 6.3.2.3 beschrieben, können die Authentifizierung und die anschließende Kommunikation zur Registrierung der Dienste und Kontexttypen über herkömmliche Protokolle erfolgen. Eine Modifikation ist nicht nötig. Der Server muss dem Kontextrouter seine Anwesenheit im Netzwerk in definierten Abständen mitteilen, da sonst die dort gespeicherten Einträge veralten und gelöscht werden. Die Anwesenheitsmitteilung sollte auch die aktuell angebotenen Dienste und die zugehörigen Kontexttypen enthalten. Dadurch können die Daten auf dem Kontextrouter aktuell gehalten werden, selbst wenn zwischenzeitlich Dienste auf dem Server aus- bzw. wegfallen oder neu hinzukommen.

Empfängt der Server eine Anfrage nach einem kontextsensitiven Dienst und erhält er mit dem zugehörigen empfangenen IP-Paket eine Sitzungsnummer und einen Identifikator, so muss er diese Werte beim Antworten der Anfrage in das zu sendende IP-Paket kopieren. Prinzipiell kann der Server auch Anfragen von Netzknoten beantworten, die ihm nicht als Kontextrouter bekannt sind. Die Kommunikation mit der kontextsensitiven Anwendung selbst wird oberhalb der Vermittlungsschicht erbracht und ist nicht Bestandteil dieser Arbeit.

6.3.4 Client

Dieser Abschnitt beschreibt die Funktionen, die mindestens vom Client erbracht werden müssen, um die Dienste der kontextsensitiven Routingarchitektur in Anspruch nehmen zu können. Die hier vorgeschlagene Syntax und Semantik zur Beschreibung der Dienste innerhalb der erweiterten RREQ- und RREP-Pakete ist dabei sehr einfach gehalten. Zum Verifizieren der Funktionen der Routingarchitektur soll die hier verwendete Lösung aber vorerst ausreichen. Bei zukünftigen Forschungen ist jedoch zu untersuchen, wie die Beschreibung der Dienste weiter optimiert werden kann.

Bevor der Client ein erweitertes AODV-RREQ-Paket zur Anfrage nach einem kontextsensitiven Dienst senden kann, benötigt er die Adresse eines Kontextrouters. Um diese zu erhalten, können prinzipiell die gleichen Mechanismen, wie sie in Abschnitt 6.3.3 beschrieben wurden, verwendet werden. Allerdings stellt hier der manuelle Eintrag der IP-Adresse eine Ausnahme (z. B. zum Festlegen eines Default-Kontextrouters) dar, da sich die Clients, wie bereits in Abschnitt 6.2.4 begründet, in der gleichen administrativen Domäne wie die Kontextrouter befinden.

Um einen kontextsensitiven Dienst zu nutzen, sendet der Client ein erweitertes AODV-RREQ an einen Kontextrouter. Dazu muss im RREQ-Header das D-Flag

gesetzt werden. Unter Nutzung des IPv6 wird beim dort verwendeten AODV die *Flood Data Option* verwendet. Beides ist notwendig, damit nur das eigentliche Ziel – nämlich der Kontextrouter – auf eine Anfrage antwortet. Die Angabe des kontextsensitiven Dienstes erfolgt in der Erweiterung des AODV-Paketes. Ein Vorschlag, wie diese Erweiterung genutzt werden kann, ist in Abbildung 6.16 dargestellt. Dabei bleibt das Format aus RFC 3561 erhalten. Im Feld *Type* wird gekennzeichnet, dass es sich um eine Anfrage nach einem kontextsensitiven Router handelt. Der Wert ist dabei frei wählbar, darf aber noch nicht für andere Erweiterungen vergeben sein. Da 1 schon für das *Hello Interval Extension Format* vergeben ist, kann für das vorliegende Szenario beispielsweise 2 verwendet werden. Mit dem Datenteil werden schließlich der Identifikator, der angeforderte Dienst, die Kontexttypen und deren Priorisierung übertragen. Dazu wird vorerst eine relativ einfache Codierung vorgeschlagen. Die ersten 4 Bytes dienen zur Übertragung des Identifikators. Dort wird der für die aktuelle Dienstanfrage der zuletzt vom Kontextrouter erhaltene Identifikator abgelegt. Das erste RREQ erhält eine 0. Die darauf folgenden 2 Bytes, beschreiben den angeforderten Dienst (*Service*). Insgesamt können somit 65535 verschiedene Dienste unterschieden werden. Im Gegensatz dazu wird beim AODV-SD die Adressierung auf die Angabe einer URL beschränkt, was aber weniger effizient und nicht so flexibel ist. Von den verbleibenden 253 Bytes werden vorerst 50 Bytes genutzt, um die Kontexttypen (*Context type (Ct)*) zu übertragen. Bestimmt werden diese durch die Position im Paket. Eine Strukturierung erfolgt nicht, d. h. mit Ct können sowohl in einem Oberbegriff zusammengesetzte als auch untergliederte Kontexttypen (z. B. Kontextsubtypen – siehe Abschnitt 4.3) angegeben werden. Zu jedem Kontexttyp wird auch dessen Priorität angegeben. Zur Priorisierung sind 3 Bit, in Abbildung 6.16 als P gekennzeichnet, vorgesehen. Damit sind 7 Prioritätsstufen möglich, wobei die Priorität mit dem Wert steigt (0 = keine Priorität; 7 = höchste Priorität). Ein Bit wird als Flag (F) verwendet, um zu kennzeichnen, dass der jeweilige Kontexttyp gewählt wurde. Damit werden pro Byte zwei Kontexttypen und deren Prioritäten beschrieben. Insgesamt können also 100 Kontexttypen kombiniert übertragen werden. Das Längenfeld (*Length*) erhält somit den Wert 56. Die Erweiterung ist insgesamt 58 Bytes groß. Natürlich kann dieses Format noch bis auf die maximal mögliche Größe erweitert werden. Das hängt von den in Zukunft tatsächlich definierten bzw. verwendeten Kontexttypen ab. Andererseits können auch beispielsweise Prioritäten für Eigenschaften (z. B. Verfügbarkeit) angegeben werden, die dann bei der Auswahl eines Servers berücksichtigt werden.

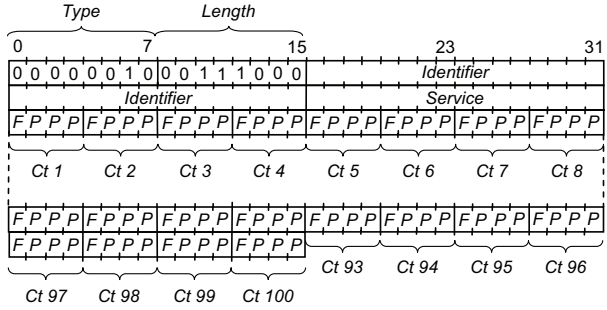


Abbildung 6.16: Kontexterweiterung des AODV-RREQ

Nach dem Versenden des erweiterten AODV-RREQ wartet der Client eine definierte Zeit (Timeout) auf das zugehörige RREP. Kommt keine Antwort zurück, kann die Anfrage mit einer endlichen Anzahl von Versuchen entsprechend der AODV-Spezifikation RFC 3561 wiederholt werden. Bleibt dies erfolglos, gilt der Kontextrouter als nicht erreichbar. Die Informationen über ihn werden gelöscht. Entweder ist nun ein alternativer Kontextrouter zu wählen oder es wird ein Solicitation-Paket gesendet, um neue Kontextrouter zu finden. Bleiben beide Varianten erfolglos, ist kein kontextsensitives Routing möglich. Es erfolgt dann eine herkömmliche Dienstanfrage ohne kontextsensitives Routingverfahren, sofern dies die kontextsensitive Anwendung unterstützt.

Erhält der Client eine Antwort auf das erweiterte RREQ, erfolgt diese in Form eines erweiterten RREP. D. h., es wird bei diesem Paketformat ebenfalls die Möglichkeit der Paketerweiterung genutzt. Wie in Abbildung 6.17 zu sehen ist, werden dort die Sitzungsnummer (*Session*), der Identifikator (*Identifier*), der Dienst (*Service*) und die vom Server unterstützten Kontexttypen (*Ct*) angegeben. Das Format der Erweiterung ist dem des RREQ-Paketes angelehnt. Das Feld *Type* ist jetzt jedoch mit dem Wert 3 belegt. Dadurch wird die Erweiterung eindeutig dem RREP zugeordnet. Natürlich kann hier aber auch ein anderer noch nicht vergebener Wert verwendet werden. Vom Datenfeld werden die ersten 2 Bytes für die Sitzungsnummer und die folgenden 4 Bytes für den Identifikator verwendet. Danach folgen 2 Bytes für den Dienst und 13 Bytes für die Kontexttypen. Die freien Stellen des letzten Bytes, das für die Kontexttypen verwendet wird, werden mit Nullen aufgefüllt, da das Format nur ganzzahlige Bytegrößen zulässt. Das Feld *Length* erhält somit den Wert 21. Insgesamt hat die Erweiterung also eine Größe von 23 Bytes. Zusätzlich befindet sich im reservierten Bereich des Paketheaders das *No Alternatives*-Flag (NA-Flag), das auf 0 gesetzt ist, solange bei der Serverwahl nicht das Ende der Serverrangliste erreicht ist. Es wird vorgeschlagen, für das NA-Flag das erste Bit im Feld Reserved des RREP-Pakets (siehe Abbildung 6.2) zu verwenden.

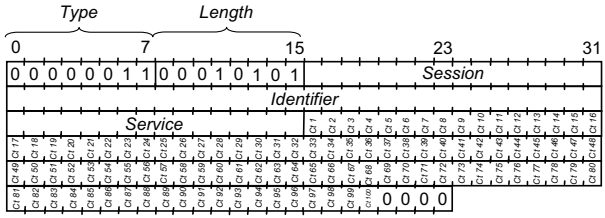


Abbildung 6.17: Kontexterweiterung des AODV-RREP

Der Identifikator dient dem Kontextrouter, wie in Abschnitt 6.3.2.6 beschrieben, zur eindeutigen Zuordnung bei der Weiterleitung der eigentlichen Daten. Da es sich dabei um die IP-Adresse des ausgewählten Servers handelt, kann der Nutzer bzw. Client entscheiden (z. B. anhand einer „Schwarzen Liste“ oder Blacklist), ob er diesen Server akzeptiert. Zusätzlich können in diesen Entscheidungsprozess auch die mitgelieferten Kontexttypen einfließen. Hierbei handelt es sich um solche Typen, die von dem ausgewählten Server und damit den dort angebotenen Dienst auch tatsächlich unterstützt werden. Daraus ergeben sich folgende Möglichkeiten einer Reaktion durch den Client:

1. Der Client lehnt den Server ab und stellt keine neue Anfrage an den Kontextrouter.

2. Der Client lehnt den Server ab und stellt eine neue Anfrage an den Kontextrouter. In der neuen Anfrage wird der Identifikator des letzten RREP mitgesendet. Dadurch kann der Kontextrouter die neue Anfrage zuordnen und einen alternativen Server anbieten. Diese Variante ist nur möglich, wenn kein NA-Flag im vorhergehenden RREP gesetzt war.

3. Der Client lehnt den Server ab und stellt eine neue Anfrage an den Kontextrouter, wobei die Kontexttypen oder deren Prioritäten in der neuen Anfrage modifiziert wurden. Es spielt hierbei keine Rolle, welchen Wert der einzusetzende Identifikator erhält. Der Kontextrouter erkennt dies als neue Anfrage.

4. Der Client akzeptiert den Server. Er verwendet den mitgelieferten Identifikator für die anschließende Kommunikation.

Als Beispiel zeigt Abbildung 6.18 ein MSC für den Fall, dass der vom Kontextrouter vorgeschlagene Server abgelehnt wurde. Daraufhin wählt der Kontextrouter einen alternativen Server aus, setzt aber das *No Alternatives*-Flag, da das Ende der Rangliste erreicht ist. Der Client akzeptiert den zweiten Server und beginnt mit diesem zu kommunizieren. Zu beachten ist, dass in der Abbildung darauf verzichtet wurde, die Sitzungsnummer mit anzugeben, da sich diese während der Kommunikationsvorgänge nicht ändert. Außerdem sei darauf verwiesen, dass die Auswahlfunktion nicht unbedingt zur Realisierung des kontextsensitiven Routing notwendig und deshalb optional ist.

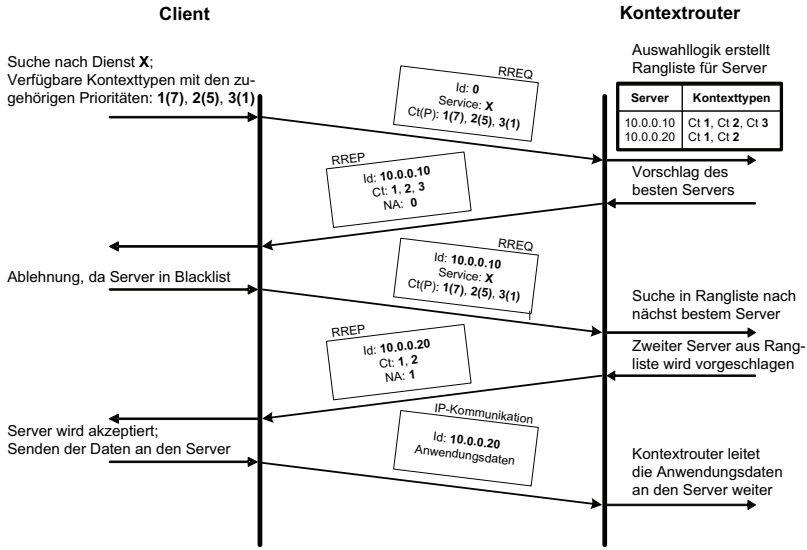


Abbildung 6.18: Auswahl eines Servers durch den Client

Die Anfrage nach einem weiteren Dienst ist unabhängig von der aktuellen Entscheidung zu jedem Zeitpunkt möglich, da vom Client durch die Angabe des Dienstes innerhalb des RREP eine eindeutige Zuordnung erfolgen kann. Erhält der Client ein RREP-Paket, bei dem eines der Felder für den Identifikator (Id) oder den Dienst den Wert 0 hat, wurde die Anfrage vom Kontextrouter abgelehnt.

Hat der Client schließlich einen Server akzeptiert, kann die eigentliche Kommunikation mit diesem beginnen. Dazu werden die IP-Pakete entsprechend Abschnitt 6.3.2.6 präpariert. Der Identifikator wird aus dem letzten RREP-Paket übernommen. Fällt während der Kommunikation der Server aus, kann der Kontextrouter die Kommunikation automatisch auf einen gleichwertigen Server umleiten. Wie im eben erwähnten Abschnitt beschrieben, ist dies nicht in jedem Fall sinnvoll. Deshalb kann der Client beim Senden des erweiterten RREQ das bereits in Abschnitt 6.3.2.6 beschriebene NR-Flag setzen und diesen Automatismus vermeiden.

Schließlich obliegt es dem Client zu entscheiden, ob eine aktuelle Anfrage durch eine Anwendung überhaupt einen kontextsensitiven Dienst verlangt. Dazu muss der Client bzw. die dort implementierte Software für das kontextsensitive Routing eine geeignete Schnittstelle bereitstellen, an der diese Informationen übergeben werden können. Damit ist der Client mit den in diesem Abschnitt unterbreiteten Vorschlägen in der Lage, einen kontextsensitiven Dienst mit Hilfe der kontextsensitiven Routingarchitektur zu finden und zu nutzen.

6.3.5 Anmerkungen zu den optionalen Funktionen

Zusätzlich zum kontextsensitiven Routing wurden auch optionale Funktionen diskutiert. Dazu gehören die Auswahl eines alternativen Servers in Verbindung mit der Nutzung einer „Schwarzen Liste“ und das Rerouting. Diese Funktionen stehen in unmittelbarem Zusammenhang, was beim praktischen Einsatz berücksichtigt werden muss. Das Rerouting wurde mit der Maßgabe konzipiert, dass der Nutzer nichts von einem Wechsel des Servers bemerkt. Damit kann er beispielsweise auch nicht bestimmen, ob der neu gewählte Server zugelassen werden soll. Deshalb kann ein Client in diesem Fall auf die Nutzung einer „Schwarzen Liste“ verzichten. Andererseits kann der Algorithmus möglicherweise auch erweitert bzw. modifiziert werden, sodass sich diese Funktionen nicht ausschließen. Dies soll aber Thema zukünftiger Forschungsarbeiten sein.

Wird auf das Rerouting verzichtet, kann auch die Sitzungsnummer weggelassen werden. Ebenso kann als Identifikator jede beliebige Zahl verwendet werden. Bei der Implementierung ist hierbei auf Eindeutigkeit zu achten. Die Identifikatoren müssen nicht auf bestimmte Server mappen, sofern auf eine Serverauswahl durch den Client verzichtet wird.

Schließlich muss auch untersucht werden, inwieweit die Funktion des Reroutings Einfluss auf die Sicherheit hat.

6.4 Bewertung

Wird das vorliegende Architekturkonzept mit den Verfahren für eine Dienstsuche aus Abschnitt 4.6 verglichen und Tabelle 4.2 entsprechend um das kontextsensitive Routing ergänzt, ergibt sich daraus Tabelle 6.6. Mit dieser Tabelle wird deutlich,

dass das kontextsensitive Routing alle Anforderungen an die Dienstsuche erfüllt. Zwar arbeitet die Architektur mit Hilfe von Verzeichnissen. Allerdings kann deren Anzahl so angepasst werden, dass die Verfügbarkeitsanforderungen in einem Ad-hoc-Netz gewährleistet werden können. Das System kann bei Bedarf über Netzgrenzen hinweg kommunizieren. Es unterstützt sowohl Infrastruktur- als auch Ad-hoc-Netze. Eine hohe Flexibilität wird durch die Unabhängigkeit zwischen Dienstsuche und Anwendungskommunikation erreicht. Außerdem ist es das einzige Verfahren, das einem Provider erlaubt, den Datenverkehr während der Dienstsuche und -kommunikation zu monitoren und zu steuern. Und schließlich unterstützt dieses Konzept die Suche nach einem Dienst unter Berücksichtigung der vom Nutzer bereitgestellten Kontexttypen, ohne dabei auf bestimmte Kontexttypen beschränkt zu sein.

Verfahren	Funktionalität	Verzeichnisdienst	Reichweite	Netzwerke	Flexibilität	Steuerbarkeit	Kontext
AODV-SD	Netzwerk	ohne	–	–	+	–	o
Bonjour	Anwendung	mit	–	o	+	–	–
DEAPspace	Anwendung	ohne	–	+	+	–	–
GSD	Anwendung	ohne	–	o	+	–	o
Jini	Anwendung	mit	+	o	o	–	o
LSD	Netzwerk	beides	–	–	+	–	–
Salutation	Anwendung	mit	+	+	+	–	o
SDP	Anwendung	mit	–	–	–	–	o
SLP	Anwendung	beides	+	o	+	–	o
UPnP	Anwendung	ohne	–	o	o	–	–
kontextsensitives Routing	Netzwerk	mit	+	+	+	+	+

Die Bewertung bezieht sich auf die in Abschnitt 4.6.2 aufgestellten Anforderungen an eine Dienstarchitektur und erfüllt diese mit: + → sehr gut; o → teilweise; – → ungenügend

Tabelle 6.6: Kontextsensitives Routing im Vergleich

Damit eignet sich das vorgestellte Konzept des kontextsensitiven Routings sehr gut für eine Dienstsuche und -kommunikation in heterogenen Netzwerkumgebungen, sofern diese auch IP unterstützen. Die Suche nach einem geeigneten Server und der zugehörigen Route erfolgt dabei in Abhängigkeit der durch den Client bereitgestellten Kontextinformationen bzw. der sich daraus ergebenden Kontexttypen.

6.5 Kapitelzusammenfassung

Das im vorliegenden Kapitel beschriebene Architekturkonzept stellt die Basis für das kontextsensitive Routing dar. Der Aufbau und die Funktionsweise wurden ausführlich beschrieben. Es erfolgte ein Vergleich verschiedener in Frage kommender Routingverfahren. Im Ergebnis erwies sich AODV als das geeignetste Routingprotokoll. Durch eine den Anforderungen an das kontextsensitive Routing entsprechende Modifikation des Protokolls konnte deshalb auf eine komplette Neuentwicklung verzichtet werden. Die Funktionen der drei Komponenten des Konzeptes – Client, kontextsensitiver Server und Kontextrouter – wurden detailliert dargestellt und die zu realisierenden Kommunikationsprozesse beschrieben. Das Kernstück der vorgeschlagenen Architektur bildet der Kontextrouter, der einerseits seine Adresse im Netz verbreiten muss, die verfügbaren kontextsensitiven Server vorhält sowie die Auswahl eines passenden Servers für den Client übernimmt und andererseits schließlich die Kommunikation zwischen Client und Server unterstützt. Durch die Weiterleitungs-

und Reroutingfunktionen sowie den Algorithmus zur Auswahl eines passenden Servers unterscheidet sich das kontextsensitive Routing von einem herkömmlichen Auskunftsdienst.

Die Anforderungen, die in diesem und den vorhergehenden Kapiteln an das kontextsensitive Routing gestellt wurden, können mit dem vorliegenden Konzept realisiert werden. Es ist so interoperabel angelegt, dass es in bestehende Netze eingepflegt werden kann. Ein Betrieb in heterogenen Netzen ist möglich. Das Konzept ist aktuell für IPv4 ausgelegt, kann aber für IPv6 übernommen werden. Die dazu notwendigen Modifikationen wurden beschrieben.

Die Funktionen werden beim kontextsensitiven Routing auf der Vermittlungsschicht realisiert. Darunter liegende Schichten werden dadurch nicht beeinflusst. Für höhere Schichten arbeitet das System ebenfalls transparent. Das schließt die kontextsensitiven Dienste mit ein. Die als Clients und Server arbeitenden Netzknoten müssen jedoch um die für das Konzept nötige Software erweitert werden. Dazu sind neben der Routingerweiterung auch Schnittstellen zur Übergabe von Dienst- bzw. Anwendungsinformationen bereitzustellen.

Vorerst sind alle Komponenten der Architektur so ausgelegt, dass sie die Mindestfunktionalität für das kontextsensitive Routing erfüllen. Zukünftige Erweiterungen sind aber möglich. Damit ist das System noch ausbaufähig. So können beispielsweise Szenarien entwickelt werden, bei denen Kontextrouter netz- oder weltweit Informationen austauschen. Am Prinzip des kontextsensitiven Routings ändert sich dadurch aber nichts.

Vollkommen unberücksichtigt sind bis dato Sicherheitsbetrachtungen. Da bei dem vorliegenden Konzept in der Regel auf bestehende Standards bzw. Spezifikationen zurückgegriffen wurde, kann an diesen Stellen von keinem erhöhten Risiko ausgegangen werden. Ob und inwieweit die kontextsensitive Routingarchitektur aber insgesamt neue Sicherheitslücken aufweist, muss noch untersucht werden. Es ist weiterhin zu erforschen, wie die Effizienz der Übertragung von Dienst und Kontexttypen durch Nutzung von Kodierungsverfahren erhöht werden kann. Teilweise wurde beim Konzept auf die Beschreibung konkreter Konfigurationswerte (z. B. Timeouts) verzichtet, da deren Größe flexibel in bestimmten Grenzen gewählt werden kann. Es sei an dieser Stelle auf die die in diesem Kapitel aufgeführten Standards verwiesen, die diesbezüglich eindeutige Vorgaben machen.

Das hier vorgestellte Konzept beschreibt alle notwendigen Funktionalitäten für das kontextsensitive Routing. Es wurde diskutiert, welche Anforderungen von den Komponenten der Architektur erfüllt werden müssen. In den folgenden Kapiteln wird gezeigt, dass dieses Konzept realisierbar ist und den gestellten Anforderungen genügt. In Kapitel 7 werden dazu zunächst Ansätze für eine simulative Umsetzung aufgezeigt.

7. Simulation

Zum Verifizieren neuer Netzwertechniken und -protokolle werden häufig Simulationen durchgeführt. Diese stellen eine kostengünstige Alternative zu einer praktischen Realisierung dar. Auch im Rahmen dieser Arbeit wurde zunächst ein solcher Ansatz gewählt, um das in Kapitel 6 vorgeschlagene Architekturkonzept bewerten zu können. In dem vorliegenden Kapitel erfolgt deshalb ein Überblick über die dazu durchgeführten Arbeiten. Nach einer Einleitung in die Thematik werden das ausgewählte Simulationswerkzeug vorgestellt und der aktuelle Stand bei der Einbindung der Routingarchitektur erörtert.

7.1 Einführung

Um die Funktionsweise einer neuen Netzwerkarchitektur verifizieren und vorführen zu können, reicht in der Regel ein praktischer Versuchsaufbau aus. Sollen jedoch Aussagen darüber getroffen werden, wie sich ein Netzwerk unter verschiedenen Verkehrsbedingungen verhält, ist in der Regel eine große Anzahl von Quellen (Teilnehmer, Clients, Server usw.) nötig. In Ad-hoc-Netzen kommt hinzu, dass diese auch noch mobil sein können. Die Komplexität solcher Szenarien lässt eine reale Umsetzung sowohl in der Entwicklungsphase als auch danach nur schwer zu. Ein praktischer Versuchsaufbau führt außerdem mit größer werdenden Netzwerken zwangsläufig zu einem steigenden Kostenaufwand. Die höher werdende Komplexität kann darüber hinaus durch den damit verbundenen Organisationsaufwand für die Messszenarien zu Fehlern führen.

Eine Alternative bieten Simulationen. Mit Hilfe von Softwarewerkzeugen wird hier das zu untersuchende Netzwerk nachgebildet. Je mehr Umgebungsparameter ein solches Werkzeug dabei berücksichtigt, desto besser entsprechen die Ergebnisse der Simulation denen der Realität. Darüber hinaus lassen sich Simulationsergebnisse genau reproduzieren, da sie nicht durch äußere Einflüsse verfälscht werden können. Es ist theoretisch möglich, reale Umgebungen bis zu einem beliebigen Grad an Detailtreue nachzubilden. Dabei ist jedoch immer abzuwägen, ob diese Detailtreue benötigt wird, da dadurch ein größerer Rechenaufwand und höhere Entwicklungskosten erzeugt werden. Am Markt sind deshalb verschiedenste Simulationswerkzeuge verfügbar, die sich auf bestimmte Netzwerke, Protokolle oder Netzwerktechniken

spezialisiert haben. Diese wurden bereits in [Born02] gegenübergestellt. Die dort für Ad-hoc-Netze als am besten geeigneten Simulatoren sind der network simulator version 2.xx (ns-2) [NS208] und die Global Mobile Information Systems Simulation Library (GloMoSim) [GloM08]. Beide werden bis heute weiter entwickelt und gelten immer noch als die bekanntesten Vertreter für Netzwerksimulatoren. Daneben hat sich in letzter Zeit mit OPNET [OPNE08] ein weiterer Simulator etabliert. Dieser ist zwar auch mit kostenlosen Forschungslizenzen erhältlich, allerdings sind im Gegenzug Forschungsergebnisse an die OPNET Technologie Inc. weiterzugeben, was zu einer gewissen Abhängigkeit führt. Der Systemkern des Simulators ist nicht quelloffen, sodass es bei Entwicklungsarbeiten unter Umständen zu Einschränkungen kommen könnte. Schließlich wird mit dem Objective Modular Network Testbed in C++ (OMNeT++) [OMNe08] ein weiterer Simulator angeboten, der für nichtkommerzielle Zwecke kostenlos genutzt werden kann. Mittlerweile hat sich dieser Simulator ebenfalls im Forschungs- und Enwicklungsbereich etabliert.

Zum Zeitpunkt der Entscheidung darüber, welcher Netzwerksimulator genutzt werden sollte, bot der ns-2 im Gegensatz zu GloMoSim mehr Flexibilität, da er neben mobilen auch verdrahtete und hybride Netzwerke unterstützte. Auf OPNET wurde aufgrund der Abhängigkeiten bezüglich des Lizenzmodells verzichtet. Dagegen war beim OMNeT zum Auswahlzeitpunkt der Funktionsumfang wesentlich kleiner als bei den anderen Simulatoren. Die Wahl fiel damit auf den ns-2. Dieser ist frei verfügbar, vollständig quelloffen und somit modifizier- und erweiterbar. Mittlerweile gibt es für diesen Simulator eine große Entwicklergemeinde und entsprechende Mailinglisten, was vor allem für die Unterstützung bei auftretenden Problemen sehr wichtig ist. Um die Einsatzmöglichkeiten des ns-2 besser darstellen zu können, wird im folgenden Abschnitt in dessen Funktionsweise eingeführt.

7.2 Die Simulationsumgebung ns-2

Der Netzwerksimulator ns-2 basiert auf der Programmiersprache C++. Der Zugriff auf die Simulationsumgebung, die Übergabe von Parametern und die Steuerung der Simulationen erfolgen mit Hilfe der *Object Tool command language* (OTcl). Die Simulationsszenarien werden in Skripten beschrieben und an den ns-2 übergeben. Es können dabei die Funktionen der verschiedenen Protokollschichten simuliert werden. Der ns-2 stellt bereits eine Reihe von Protokollen zur Verfügung. Unterstützt werden unter anderem auch Routingprotokolle für Ad-hoc-Netze – wie beispielsweise das AODV.

Der ns-2 wurde für Unix/Linux-Systeme entwickelt. Er kann aber auch mit Hilfe von Unix-Emulatoren (z. B. cygwin [cygw08]) unter Windows verwendet werden. Die Bedienung bzw. der Aufruf des ns-2 erfolgt textzeilenbasiert. Die Ergebnisse der Simulation werden in einer Trace-Datei abgelegt. Zur Auswertung bietet die ns-2-Umgebung zwei Werkzeuge. Mit dem network animator (nam) können die Simulationen visualisiert und mit Xgraph die Verkehrsdaten dargestellt werden. Darüber hinaus bieten andere Entwickler weitere Werkzeuge zur anschaulichen Darstellung von Ergebnissen an. Ein bekanntes Beispiel hierfür ist Trace Graph [Male08]. Zur Visualisierung gibt es neben dem nam weitere Hilsmittel, die spezielle Bedürfnisse berücksichtigen. Abbildung 7.1 zeigt die Darstellung eines Simulationsverlaufes mit Hilfe des Programms viewtraffic. Dabei handelt es sich um eine in [Born02] vorgestellte Softwarelösung, die bei einer Simulation die aktuell zwischen den Netzknoten vorhandenen Routen ausgibt.

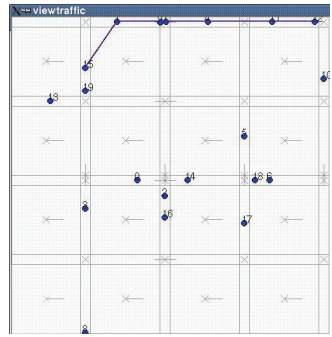


Abbildung 7.1: Visualisierung einer dem Manhattan-Modell angelehnten Simulation mit viewtraffic [Born02]

7.3 Simulationsszenarien

Simulationen werden insbesondere für Perfomanceuntersuchungen eingesetzt. Dabei ist für die Gestaltung des Simulationsszenarios das jeweilige Ziel der Untersuchung zu berücksichtigen. So können allgemeine Aussagen zu Verkehrscharakteristiken innerhalb bestimmter Umgebungsbedingungen getroffen werden. Es können aber auch Szenarien entwickelt werden, die versuchen eine reale Situation möglichst genau zu repräsentieren. Für Ad-hoc-Netze würde dies bedeuten, dass beispielsweise die Bewegungen von mobilen Netzknoten entsprechend den Bewegungen realer Teilnehmer nachgebildet werden. Dazu müssen geeignete Bewegungsmodelle erstellt und in den Simulator eingepflegt werden. In [Born02] wurden diesbezüglich verschiedene Modelle aus [Jugl00] untersucht. Die daraus resultierenden Ergebnisse führten zu der Aussage, dass die allgemeingültigen Bewegungsmodelle aus dem zellenbasierten Mobilfunkbereich wie beispielsweise Fluid-Flow-Modelle [ThGM88] oder das Hong/Rappaport-Modell [HoRa86] gar nicht oder nur schlecht in Simulationen für Ad-hoc-Netzen verwendet werden können. Andere Modelle aus diesem Bereich wie z. B. die Modelle nach Guérin [Guér87], nach Zonoozi [ZoDa97] oder solche, die auf der Brownschen Bewegung beruhen [LeRo97, LeRo98], lassen sich dagegen sehr gut nutzen. Dies gilt nach [Born02] ebenso für das in [ETSI98] beschriebene Manhattan-Modell für UMTS (siehe Abbildung 7.1) oder Gebietsmodelle, bei denen sich die Netzknoten entsprechend dem Gebiet, in dem sie sich befinden, verhalten.

Es ist dabei zu beachten, dass jedes Modell nur eine begrenzte Aussagekraft besitzt und in der Regel nicht allgemeingültig angewendet werden kann. Für eine Simulation muss dann das Bewegungsmodell gewählt werden, welches den realen Gegebenheiten am besten entspricht. Die Ergebnisse sind also immer mit Rücksicht auf die jeweiligen Simulationsparameter zu bewerten. Neue Bewegungsmodelle können zwar mit einer beliebigen Genauigkeit an eine bestimmte Umgebung angepasst werden. Allerdings würden die Simulationen dann trotz des großen Aufwandes nur für dieses eine Gebiet gelten. Die anderen Modelle liefern dagegen nicht so genaue Ergebnisse, können dafür aber flexibel eingesetzt werden.

Die Bedeutung von Bewegungsmodellen wird auch durch [CoGi07] bestätigt. Danach gehen viele Simulationen nur von zufälligen Bewegungen aus, die jedoch nirgendwo

in der Praxis zu finden sind. Dementsprechend sind dann auch die Simulationsergebnisse zu bewerten. Zusammenfassend betrachtet, bilden die in [Born02] vorgestellten für Ad-hoc-Netze geeigneten Bewegungsmodelle eine gute Basis für Simulationen, die das Verhalten der in dieser Arbeit vorgestellten kontextsensitiven Routingarchitektur widerspiegeln können.

7.4 Aktueller Stand

Mit der praktischen Realisierung zu [Pasc05] wurde erstmals das im ns-2 integrierte AODV für das kontextsensitive Routing modifiziert. In diesem Ansatz wurden die Kontexttypen allerdings noch, wie in Abbildung 7.2 gezeigt, im reservierten Bereich des AODV-RREQ-Pakets angeordnet. Hier wird genau ein Byte genutzt, um die Kontexttypen (Ct) zu übertragen. Somit können maximal 8 verschiedene Kontexttypen gleichzeitig oder aber 255 verschiedene Kontexttypen einzeln übertragen werden. Da zum Zeitpunkt der Implementierung in den ns-2 noch keine Entscheidung bezüglich der Methodik zur Adressierung der Kontextrouter getroffen war, wurde deshalb der folgende Ansatz verwendet: Die Anfrage eines Clients erfolgt mit einer Adresse, die keinem Knoten zugeordnet ist. Dadurch wird die Anfrage durch das gesamte Netzwerk geführt und schließlich nur beim Knoten, der Informationen zu diesem Kontexttyp besitzt (dem Kontextrouter) ausgewertet. Ein Dienst wurde hier noch nicht spezifiziert und auch die Kommunikation zwischen Client und Server blieb unberücksichtigt. Dies wurde allerdings auch nicht benötigt, da das Ziel darin bestand, Aussagen über das Verkehrsverhalten beim Routingprozess treffen zu können.

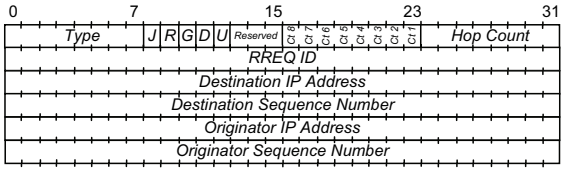


Abbildung 7.2: Format des erweiterten RREQ im ns-2

Als problematisch erwies sich bei der Implementierung die Tatsache, dass innerhalb eines Simulationsszenarios lediglich Netzknoten desselben Typs verwendet werden konnten. D. h., dass Simulationen entweder nur mit Kontextroutern, die jeweils die gleichen Informationen besitzen, oder nur mit herkömmlichen AODV-Knoten möglich waren. Ein Netzwerk mit verschiedenen Knoten konnte nicht realisiert werden. Entsprechend wurde auf Simulationen großer Netzwerke verzichtet, da deren Aussagekraft keine Bedeutung besessen hätte. Es konnte jedoch in [Pasc05] das korrekte Verhalten eines als Kontextrouter implementierten Knotens nachgewiesen werden. Dieser antwortet auf erweiterte RREQs, sofern in der Anfrage dem Router bekannte Kontexttypen enthalten sind. Werden dagegen Pakete empfangen, die unbekannte oder keine Kontexttypen übertragen, ist das Verhalten des Kontextrouters entsprechend dem AODV und es erfolgt eine Weiterleitung an benachbarte Netzknoten. Aus diesen ersten Simulationen wurden wesentliche Erkenntnisse für die Entwicklung des in Kapitel 6 beschriebenen Konzepts gewonnen.

Inzwischen ist in [Henn06] analysiert worden, warum in [Pasc05] nur Knoten des gleichen Typs simuliert werden konnten. Die Ergebnisse führten zu einer Implementierung, die Simulationen mit verschiedenen Knotentypen unterstützt. Somit lassen sich zukünftig die AODV-Erweiterungen des aktuellen Architekturkonzeptes analog zu den bereits in den ns-2 eingepflegten Varianten implementieren.

Während der Arbeiten zu [Pasc05] wurden zusätzlich in [Schm05] Untersuchungen dazu durchgeführt, welche Möglichkeiten der Adressierung eines Kontextrouters bestehen (siehe auch Abschnitt 6.3.1) und wie die ausgewählte Lösung in den ns-2 implementiert werden kann. Es wird dort vorgeschlagen, die Kontextrouter mit Hilfe von Multicast-Adressen zu adressieren. Dazu musste das bereits im ns-2 implementierte AODV um die Funktionalität des Multicast erweitert werden. Hierbei wurde auf die Implementierung aus [ZhKu04] zurückgegriffen. Zum Zeitpunkt der Entscheidung für die Multicastadressierung erwies sich diese Lösung mit Rücksicht auf die bis dato vorhandenen Entwicklungsergebnisse aus [Pasc05] und dem damaligen Konzeptstatus als beste Variante. Die Simulationsergebnisse führten jedoch dazu, eine andere Adressierungsvariante für das Konzept zu verwenden. Diese ist in Abschnitt 6.3.1 beschrieben und begründet.

Sowohl die Arbeit von [Pasc05] als auch diejenige von [Schm05] vermittelten einen Eindruck der Komplexität, die sich durch die Implementierung einer kontextsensitiven Routingarchitektur in den ns-2 ergibt. Die Entwicklung des Architekturkonzeptes erfolgte parallel zu diesen Arbeiten, sodass Schwachstellen aufgedeckt und bei der weiteren Entwicklung berücksichtigt werden konnten.

Zu Beginn der Entwicklung der Simulationsumgebung war der dafür notwendige zeitliche Aufwand nur schwer abschätzbar. Deshalb wurde frühzeitig damit begonnen, auch Möglichkeiten einer praktischen Umsetzung zu untersuchen (z. B. [Kram05, Renh06]). Während der Arbeiten an der Simulationsumgebung stellte sich dann heraus, dass vor allem Versionskonflikte beim ns-2 deren Entwicklungsfortschritt behinderten. Beispielsweise basierte die bei [Schm05] verwendete AODV-Multicastimplementierung auf einer älteren Softwareversion des ns-2 als die Arbeit von [Pasc05]. Wegen Kompatibilitätsproblemen konnte deshalb die Arbeit von [Schm05] nicht in diejenige von [Pasc05] eingebunden werden. Weiterführende Arbeiten wären mit einem erhöhten Zeitaufwand verbunden gewesen, um die bestehenden Probleme zunächst zu beseitigen und zukünftig zu verhindern. Ein Erfolg der Arbeiten war nicht garantiert. Die Ergebnisse von [Kram05] und [Renh06] wiesen dagegen auf sehr gute Erfolgsaussichten für eine praktische Realisierung hin. Im direkten Vergleich mussten schließlich die Arbeiten an den Simulationen als nicht zielführend eingestuft werden. Dieser Erkenntnis folgend, wurde die Entscheidung getroffen, den Schwerpunkt zukünftig auf eine praktische Realisierung zu legen. Somit konnte gewährleistet werden, dass die Funktionen des Architekturkonzeptes innerhalb eines vertretbaren Entwicklungszeitraumes verifizierbar sein würden.

Aus aktueller Sicht wird diese Entscheidung durch [CoGi07] bestätigt. Hier wurden die Simulationsergebnisse von ns-2, GloMoSim und OPNET miteinander verglichen. Die Simulatoren offenbarten dabei sowohl bedeutende quantitative (Größe von absoluten Werten) als auch qualitative (Verhalten) Unterschiede. Dies wird in [CoGi07] unter anderem auf die unterschiedlichen in den Simulatoren verwendeten Modelle für die Bitübertragungsschicht zurückgeführt. Dementsprechend besteht bei den Simulatoren selbst noch Entwicklungsbedarf. Sollten trotzdem damit Simulationen durch-

geführt werden, müssen deren Ergebnisse kritisch hinterfragt werden. Hier kann auch eine praktische Realisierung helfen, zukünftige Simulationen bzw. deren Ergebnisse zu verifizieren.

Wie später in Kapitel 8 erläutert wird, fiel die Entscheidung bei der praktischen Realisierung des Kontextrouters auf das Softwarewerkzeug Click. Diese Entscheidung erwies sich auch deshalb als vorteilhaft, da Click eine Schnittstelle für den ns-2 bereithält. Das stellt eine gute Voraussetzung für zukünftige Entwicklungen auf dem Gebiet der Simulationen dar, weil damit praktisch realisierte Router in den ns-2 integriert werden können.

7.5 Kapitelzusammenfassung

Um die Vorteile des kontextsensitiven Routings auch in großen Netzwerken nachweisen zu können, sind Untersuchungen mit Hilfe von Simulationen notwendig. Hierzu ist einführend auf die Bedeutung von Bewegungsmodellen und deren Einfluss auf die Simulationsergebnisse eingegangen worden. Es wurde darüber hinaus beschrieben, warum als Simulationswerkzeug der ns-2 ausgewählt wurde. Spezielle Eigenschaften dieses Simulators wurden erörtert. Es sind Implementierungen vorgestellt worden, die als Basis für die Simulation der Kommunikationsprozesse in der Routingarchitektur dienen können.

Aus den in diesem Kapitel beschriebenen Gründen ist entschieden worden, das Architekturkonzept für das kontextsensitive Routing über eine praktische Realisierung zu verifizieren. Im Falle zukünftiger Simulationen ist unbedingt auch der Entwicklungsstand des Simulators zu berücksichtigen. Wie gezeigt wurde, gibt es hier gerade bei der Realisierung der Bitübertragungsschicht noch Defizite. Eine Aussage darüber, inwieweit zukünftige Simulationsergebnisse als repräsentativ eingestuft werden können, ist durch den direkten Vergleich mit praktisch realisierten Szenarien möglich. Dazu bietet sich beispielsweise der im Rahmen dieser Arbeit entwickelte Demonstrator für das kontextsensitive Routing an. Die für diesen entwickelten Implementierungen beschreibt das folgenden Kapitel 8.

8. Praktische Realisierung des Konzeptes

Während sich Simulationen für verkehrstheoretische Untersuchungen anbieten, eignen sich praktische Realisierungen im Rahmen eines Demonstrators insbesondere für den Nachweis der Funktionsfähigkeit eines Architekturkonzeptes. Der Vorteil besteht darin, dass sich Stärken und Schwächen für den produktiven Einsatz eines Systems besser abschätzen lassen. Eventuelle Interoperabilitätsprobleme zwischen Protokollen und Techniken können direkt aufgedeckt werden. Mängel einer Architektur sind sofort sichtbar. Lösungsansätze zu deren Behebung können direkt am System entwickelt und getestet werden. Zusätzlich sind auch Aussagen über den Aufwand der Implementierung unter Verwendung diverser Netzwerktechniken möglich. Darüber hinaus steht eine Basis zum Verifizieren zukünftiger Erweiterungen und Innovationen bereit.

In den folgenden Abschnitten wird beschrieben, wie die Netzknoten für das Architekturkonzept des kontextsensitiven Routings praktisch realisiert werden können. Verschiedene Möglichkeiten der Implementierung werden vorgestellt und verglichen. Es erfolgen eine Beschreibung des aktuellen Entwicklungsstandes der einzelnen zum Architekturkonzept gehörenden Komponenten sowie der Möglichkeiten zukünftiger Erweiterungen.

8.1 Einführung

Wie in Kapitel 6 beschrieben, sind Kontextrouter, Client und Server die wesentlichen Bestandteile des kontextsensitiven Routings. Basierend auf den im Konzept spezifizierten Anforderungen, wurden diese Komponenten praktisch realisiert. Ziel dabei war, Implementierungen zu entwickeln, die so modular und flexibel gestaltet sind, dass ein möglichst breites Spektrum an Tests und Messungen unterstützt wird. Die Implementierungen können als Middleware betrachtet und wie in Abbildung 8.1 innerhalb eines Systems eingeordnet werden. Sowohl Kontextrouter als auch Server und Client können auf die gleiche Hardware aufsetzen und sind deshalb einfach zwischen den in einem Netzwerk beteiligten Knoten portierbar. Die Anwendungen

wiederum, die auf die jeweilige Implementierung zugreifen, können dem Knotentyp entsprechend spezialisiert sein. In Abbildung 8.1 soll dies die gestrichelte Linie verdeutlichen.

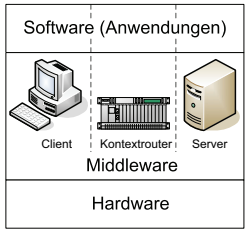


Abbildung 8.1: Softwareimplementierungen

Die Implementierungen selbst sollen dem Nutzer Möglichkeiten zur Konfiguration und Anpassung an die jeweilige Einsatzumgebung bieten. Dies trifft im Besonderen auf den Kontextrouter zu, da er den Mittelpunkt des kontextsensitiven Routings bildet. Um Zusammenhänge während einer Kommunikation bzw. das Verhalten der Netzwerkumgebung besser verstehen zu können, müssen deshalb Schnittstellen vorgesehen werden, die die Änderung von einzelnen Konfigurationsparametern wie beispielsweise Timeouts ermöglichen. Dadurch sind auch bei zukünftigen Arbeiten Optimierungen bzw. eine Anpassung des Netzwerkverhaltens möglich.

Weil das Konzept allgemein die Unterstützung von IP-Netzwerken fordert, wurden die Implementierungen so realisiert, dass möglichst unterschiedliche Netzwerktechniken unterstützt werden. Mindestens sind aber im Rahmen eines IP-Demonstratornetzwerkes ein LAN als Infrastrukturnetz und ein Ad-hoc-Netz notwendig, damit die im Konzept beschriebenen Funktionen nachgewiesen werden können. Für das Ad-hoc-Netz wird dabei vorzugsweise WLAN als Technik verwendet, was sich nach Kapitel 2 als das dafür am besten geeignete erwies.

Das Ad-hoc-Netz bildete bei der Umsetzung des Konzeptes eine besondere Herausforderung, da es gegenwärtig leider noch sehr wenige Implementierungen gibt, die entsprechende Protokolle unterstützen. Erste Erfahrungen mit Ad-hoc-Netzen wurden in [Kram05] gesammelt. Dort wurden verschiedene Implementierungen für Routingprotokolle getestet. Es erfolgte eine Gegenüberstellung von sowohl am Markt verfügbaren Implementierungen als auch von Softwarewerkzeugen zur Realisierung von Ad-hoc-Netzen. Das gestattete eine erste Vorauswahl geeigneter Software, auf der die folgenden Entwicklungen aufbauen konnten.

8.2 Kontextrouter

Die praktische Realisierung des Routers erfolgte gemäß den im Konzept spezifizierten Anforderungen. Die Konfigurations- bzw. Protokollparameter orientieren sich nach den entsprechenden RFCs. Des Weiteren wurde Wert darauf gelegt, dass solche Parameter mit Rücksicht auf einen möglichst flexiblen Einsatz in Testumgebungen modifiziert werden können.

8.2.1 Möglichkeiten der Umsetzung

Um die geforderten Funktionen des kontextsensitiven Routers umsetzen zu können, bedurfte es einer detaillierten Untersuchung der aktuellen Implementierungen. Entsprechende Recherchen wurden in [Renh06] durchgeführt. Die dabei ermittelten Routerarchitekturen können nach ihrer Implementierung Hardware- und Softwarerealisierungen gegliedert werden. Ist die Architektur der jeweiligen Implementierung bekannt oder sind die benötigten Schnittstellen offen gelegt, können diese für Entwicklungszwecke modifiziert oder auch erweitert werden.

Stellvertretend für Hardwarerouter sollen die Marktführer im Enterprise-Segment – Cisco und Juniper – kurz betrachtet werden. Diese besitzen firmenspezifische Betriebssysteme (Cisco: *Internetwork Operating System* (IOS); Juniper: *Juniper Operating System* (JunOS)). Die darauf basierenden Router haben zwar auf Seiten der Performance Vorteile, leider besteht aber kaum eine Möglichkeit, auf deren Implementierung zuzugreifen. Die Betriebssysteme der beiden Hersteller sind nicht quelloffen und bieten keine Schnittstellen für einen Zugriff. Damit wird dem Entwickler die Möglichkeit der Modifikation bzw. der Implementierung neuer Funktionen verwehrt.

Daneben gibt es auch Hardwarerouter (z. B. WRT54GL von Linksys), die mit einem quelloffenen Betriebssystem ausgerüstet sind bzw. werden können. Die Firmware solcher Router basiert meistens auf Linux. Stellvertretend seien hier DD-WRT [wrt008] und OpenWRT [owrt08] genannt, die schon an der Bezeichnung WRT erkennen lassen, dass die Implementierungen ursprünglich für Router der Firma Linksys entwickelt wurden. Mittlerweile werden aber auch Router anderer Hersteller von diesen Betriebssystemen unterstützt. Modifikationen können an einer solchen Firmware zwar erfolgen, jedoch kommen die Router aus dem Heimbereich, was mit einer relativ geringen Speichergröße und Rechenleistung verbunden ist. Das kann ggf. die Erweiterung bestehender und die Entwicklung neuer Funktionen oder auch Protokolle einschränken.

Dagegen scheinen komplette Softwarelösungen während der Entwicklungsphase neuer Systeme besser geeignet zu sein. Herkömmliche Rechnersysteme können hierbei als Router verwendet werden. Dies wird in der Regel von den aktuellen Betriebssystemen unterstützt. Vor allem UNIX-Derivate bzw. Linux spielen hierbei eine große Rolle, da dafür der Quellcode offen gelegt ist. Das Konfigurationswerkzeug iptables als bekanntester Vertreter unterstützt beispielsweise umfangreiche Filter- und Manipulationsfunktionen. Des Weiteren sind in den Betriebssystemen teilweise Routingprotokolle wie das *Routing Information Protocol* (RIP), *Open Shortest Path First* (OSPF) o. ä. implementiert. Zu beachten ist auch bei einer Softwarelösung, dass hier letztlich Hardware für die Netzwerkkommunikation benötigt wird. Bei gewöhnlicher PC-Technik als Hardwareplattform können sich deshalb die für die Anbindung von Netzwerkkarten verwendeten Bussysteme durchaus als Flaschenhals erweisen. So unterstützt beispielsweise der *Peripheral Component Interconnect*-Bus (PCI-Bus) mit einer Taktrate von 33,33 MHz und einem Übertragungsvolumen von 32 Bit pro Takt eine maximale Transferrate von 133 MByte/s. Dies könnte sich beispielsweise auf den Betrieb einer Gigabit-Ethernetkarte mit 4 Ports (z. B. Intel PWLA8494GT) negativ auswirken. Aktuell gibt es aber auch schon Bussysteme mit einer maximalen Transferrate von bis zu 8 GByte/s (PCI Express 2.0 x16) [Wind07]. Wichtig für den Einsatz in einem Produktionsnetz ist also eine ausreichende Dimensionierung der Hardware.

Auch Microsoft gestattet den Zugriff auf seine Betriebssysteme, um Routingfunktionen nutzen bzw. diese modifizieren zu können. Allerdings ist dort der Quellcode des Kernels nicht offen. Soll ein Protokoll oder Router von Grund auf neu programmiert werden, ist jedoch ein Zugriff auf die Kernelfunktionen notwendig. Sofern Microsoft entsprechende APIs anbietet, sind diese im Allgemeinen eingeschränkt zugänglich, kostenpflichtig und/oder unterliegen ggf. zusätzlichen Nutzungsbeschränkungen.

Unabhängig davon, ob der Quellcode frei zugänglich ist oder nicht, erweist sich die Programmierung des Kernels sowohl für Windows als auch für Linux als sehr komplex. An diesem Punkt setzt das Softwarewerkzeug Click an. Dieses ist speziell an die Bedürfnisse der Entwickler von Routersoftware angepasst. Es vereint die Vorzüge der bereits beschriebenen Softwarelösungen und ist für die Entwicklung von Routingprotokollen bzw. Routerfunktionen viel komfortabler, da der direkte Zugriff auf die Kernelfunktionen eines Betriebssystems entfällt. Eine Beschreibung von Click erfolgt in Abschnitt 8.2.2.

System	Konzept	Lizenz	Erweiterbarkeit	Komfort
IOS	Firmware	Cisco	–	–
JunOS	Firmware	Juniper Networks	–	–
DD-WRT, OpenWrt	Firmware	*Open Source*	o	+
Linux	PC-Betriebssystem	*Open Source*	+	+
Windows	PC-Betriebssystem	Microsoft	+	o
Click	Sofware (Linux)	*Open Source*	+	++

Tabelle 8.1: Vergleich typischer Routerkonzepte

Tabelle 8.1 stellt die verschiedenen Routerkonzepte noch einmal gegenüber. Im Einzelnen werden folgende Eigenschaften verglichen:

Konzept beschreibt, auf welcher Ebene die Funktionen des Routers initiiert werden.

Lizenz gibt Auskunft über die Nutzungsrechte. „Open Source“ bezeichnet freie Software, deren Quelltext verfügbar ist und an der Erweiterungen bzw. Modifikationen erlaubt sind.

Erweiterbarkeit bewertet, inwieweit das Routingsystem erweitert bzw. modifiziert werden kann. Diesbezüglich werden (–) für keine, (o) für eingeschränkte und (+) für weitreichende Möglichkeiten unterschieden.

Komfort stellt ein Maß für die Entwicklerfreundlichkeit dar. Mit (–) werden die Systeme bewertet, die keinen Zugriff erlauben. (o) kennzeichnen solche Systeme, deren Architektur umfangreiche Kenntnisse voraussetzen bzw. für die nur kommerzielle APIs als Werkzeug zur Verfügung stehen. Sofern systemeigene Werkzeuge zum Zugriff auf das Routingsystem bereitgestellt werden, wurde dies mit (+) bewertet. Die Routersoftware Click wurde deshalb mit (++) bewertet, weil sie durch ihr modulares und objektorientiertes Konzept den Ansprüchen der Routerentwicklung am meisten entspricht.

Tabelle 8.1 zeigt deutlich die Vorzüge von Click. Es ist sehr flexibel und bietet im Vergleich mit den anderen Varianten die komfortabelsten Schnittstellen für Entwickler. Trotzdem ist durch die Quelloffenheit ein beliebig tiefer Zugriff sowohl in die Routingsoftware als auch in das Betriebssystem möglich. Damit stellt Click den vielversprechendsten Ansatz dar, um einen Router bzw. ein neues Routingprotokoll zu realisieren.

8.2.2 Click

Click [CMRP07] ist ein Entwicklungswerkzeug, das auf der Programmiersprache C++ basiert. Es erlaubt durch seine modulare und objektorientierte Struktur, Routingprotokolle und Router mit relativ geringem Aufwand umzusetzen und zu entwickeln. Die Software arbeitet unter Linux und kann sowohl im *Userlevel*-Modus als auch im *Kernel*-Modus betrieben werden. Der *Userlevel*-Modus dient eigentlich dazu, Entwicklern eine Test- und Entwicklungsumgebung für Routersoftware zu liefern, während der fertiggestellte Router beim Einsatz im Produktionsnetz im *Kernel*-Modus arbeitet. In diesem Modus werden die betriebssystemspezifischen Paketoperationen durch in Click implementierte Mechanismen ersetzt. Da die Funktionalitäten direkt in den Kernel compiliert werden, erhöht sich auch die Leistungsfähigkeit des Routers bezüglich der Datenrate im Netzwerk. Andererseits ist dieser Modus für eine Test-/Entwicklungsumgebung unflexibel, da der Kernel nach jeder Änderung der Routerkonfiguration erst neu compiliert werden muss.

Der modulare Aufbau von Click basiert auf der Verwendung so genannter Elemente. Diese besitzen bestimmte Funktionen sowie Ein- und Ausgänge. Elemente können modifiziert oder auch neu erstellt werden. Sie sind über Ein- und Ausgänge (Ports) miteinander verbunden. Die Kommunikation erfolgt dann mittels Pakete, die entsprechend der Ports der Elemente mit oder ohne Aufforderung gesendet werden können. Elemente besitzen dazu Ports für *Push*-, *Pull*- oder *Agnostic*-Verbindungen. Während die Pakete bei *Push*-Verbindungen ohne explizite Aufforderung versendet werden, erfolgt dies bei *Pull*-Verbindungen nur mit einer Aufforderung an die Quelle. Die *Agnostic*-Ports unterstützten zwar beide Arten, bei der Initialisierung wird jedoch entsprechend den Randbedingungen nur eine ausgewählt. So werden in dem Beispiel aus Abbildung 8.2 mit dem Element *FromDevice(eth0)* an der Netzwerkschnittstelle „eth0“ eintreffende Pakete ausgelesen und an das Element *Counter* weitergeleitet. Dort werden dann die Pakete gezählt und schließlich mit Hilfe des Elementes *Discard* verworfen. *Counter* besitzt zwei *Agnostic*-Ports, die entsprechend einer *Push*-Verbindung arbeiten, weil sowohl der Ausgangsport von *FromDevice(eth0)* als auch der Eingangsport von *Discard* für eine *Push*-Verbindung ausgelegt sind. Detailliertere Informationen und eine sehr gute Einführung in die Thematik erfolgt in [Wenz06]. Hier ist auch der aktuelle Entwicklungsstand der Software dargestellt.

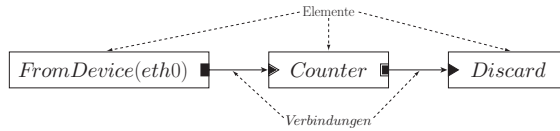


Abbildung 8.2: Beispielkonfiguration mit Click [Wenz06]

8.2.3 Implementierung

Eine erste Realisierung des Kontextrouters mit Hilfe von Click erfolgte in [Wenz07a]. Der dort umgesetzte Routingalgorithmus baut auf einer AODV-Implementierung von [Brae05] auf. Die Realisierung des Kontextrouters basiert auf dem Konzept in Kapitel 6. Neben den Erweiterungen für das kontextsensitive Routing werden auch weiterhin das AODV nach RFC 3561 [BRPD03] und IPv4 nach RFC 791 [Post81a] jeweils ohne Erweiterungen unterstützt. Der Einsatz in herkömmlichen IP-Netzen ist deshalb unproblematisch, wie in Abschnitt 9.3.4 noch nachgewiesen wird.

8.2.3.1 Funktionen

In der aktuellen Version sind für das kontextsensitive Routing folgende Funktionen realisiert:

- Generieren von Advertisements
- Antworten auf Solicitations
- Auswerten erweiterter AODV-RREQ
- Auswahl eines geeigneten Servers
- Auswahl alternativer Server
- Antwort mit erweitertem AODV-RREP
- Unterstützung des Rerouting
- Registrierung für Server incl. Update-Prozeduren
- Unterstützung der Kommunikation zwischen Client und Server (Weiterleitung von herkömmlichen und der für das kontextsensitive Routing modifizierten IP-Pakete)

Da die aufgelisteten Funktionen dem Konzept aus Kapitel 6 entsprechen, wird hier nicht noch einmal darauf eingegangen. Für Details wird auf [Wenz07a] verwiesen. Es werden im Folgenden lediglich solche Eigenschaften erläutert, die über das Konzept hinaus bei der praktischen Umsetzung berücksichtigt werden mussten.

Algorithmen für die Serverauswahl

Beim Rerouting wird eine sogenannte dynamische Rangliste verwendet. D. h., dass bei jedem Reroutingversuch eine neue Liste mit aktuellen Servern zusammengestellt wird. Dabei werden als Alternative nur Server ausgewählt, die genau die Menge der geforderten Kontexttypen unterstützen oder sich höchstens um einen bestimmten Maximalwert von dieser Menge unterscheiden.

Registrierung der Dienste

In Abschnitt 6.3.2.3 wurde allgemein auf die Anforderungen bei der Registrierung von Diensten auf dem Kontextrouter eingegangen und vorgeschlagen, zur Authentifizierung ein Challenge-Response-Verfahren zu nutzen. Für die Kommunikation zwischen Client und Server wird nun die *Secure-Shell* (SSH) verwendet. Nach der erfolgreichen Authentifizierung kann der Server seine Dienste und die zugehörigen unterstützten Kontexttypen via Datei übermitteln. Im konkreten Fall erfolgen Authentifizierung und verschlüsseltes Übertragen mittels zur SSH gehörendem Kopierbefehl `scp`. Es wird dabei mit asymmetrischer Verschlüsselung gearbeitet. Das Schlüsselpaar (öffentlicher und privater Schlüssel) wird vom Dienstanbieter (Server) generiert und bei beiden Kommunikationspartnern hinterlegt. Der öffentliche Schlüssel wird auf dem Kontextrouter und der private auf dem Server gespeichert. Beim Registrierungsvorgang kann der Server mit Hilfe des öffentlichen Schlüssels die Identität des Servers feststellen. Eine Passwortabfrage erfolgt nicht. Damit diese Prozedur funktioniert, muss ein Login auf dem Kontextrouter angelegt werden. Der Pfad, in dem die Registrierungsdatei abgelegt werden soll, ist über die Paramter des Kontextrouters (siehe Anhang A) konfigurierbar.

Für die Übertragung der Registrierungsdatei gelten folgende Vereinbarungen. Der Dateiname wird aus der IP-Adresse des Servers und der Endung „.list“ gebildet. Die Angabe der IP-Adresse erfolgt, indem die vier Bytes der Adresse entsprechend der *Network Byte Order* umgestellt und deren Binärwerte zu einer Zahl zusammengefügt werden. Die sich daraus ergebende eindeutige Dezimalzahl dient schließlich der Bezeichnung der Datei. Das Format wurde deshalb gewählt, weil es die Weiterverarbeitung im Router erleichtert. Die IP-Adresse 10.0.0.30 würde somit beispielsweise durch 503316490 dargestellt werden. Der Inhalt der Datei besteht ebenfalls aus Dezimalzahlen. Dabei repräsentiert jede Zeile einen Dienst. Folgendes Beispiel einer solchen Datei soll das verdeutlichen:

```
1 4 5 7 8;
3 1 5 6;
12 1 2 5;
27 2 57 79 99;
```

Die erste Zahl kennzeichnet hierbei den Dienst und die folgenden die vom Dienst in aufsteigender Reihenfolge unterstützten Kontexttypen. Es werden somit vier Dienste (1, 3, 12 und 27) angeboten. Dienst 1 unterstützt dabei die Kontexttypen 4, 5, 7 und 8, Dienst 3 die Kontexttypen 1, 5 sowie 6 usw. Alle Zahlen müssen durch Leerzeichen voneinander getrennt werden. Entsprechend den für das kontextsensitive Routing erweiterten AODV-Paketen werden 65535 verschiedene Dienste und 100 Kontexttypen unterstützt. Jede Zeile ist mit einem Semikolon abzuschließen.

8.2.3.2 Besonderheiten

Da das AODV als Protokoll nicht für Infrastrukturnetze entwickelt wurde, sind für die Implementierung einige Besonderheiten zu berücksichtigen.

Erweiterte RREQ- und RREP-Pakete

Die Knoten des IP-Infrastrukturnetzes können keine Sequenznummern für RREQ- und RREP-Pakete vergeben oder die eines betreffenden Zieles ermitteln. Das ergibt

sich aus der Tatsache, dass diese AODV-Pakete hier lediglich zur Dienstsuche genutzt werden. Eine Routensuche entfällt, da sich alle an der Kommunikation beteiligten Geräte in einem Subnetz befinden. Da die Paketformate für das erweiterte AODV beibehalten werden, erhalten die entsprechenden Felder für die Sequenznummern den Wert Null. Die Sequenznummer dient eigentlich dazu, Schleifen bei Routenanfragen zu vermeiden. Da die Anfrage aber nur innerhalb eines IP-Subnetzwerkes stattfindet, kann dort auf diese Funktion verzichtet werden. Aus dem gleichen Grund werden für solche Netzwerke beim erweiterten RREP-Paket die Werte für *Prefix Size*, *Hop Count* und *Destination Sequence Number* auf den Wert Null gesetzt.

Die Implementierungen verwenden an den Netzwerkinterfaces für die IP-Infrastrukturnetze außer den erweiterten RREQ- und RREP-Paketen keine weiteren Funktionen des AODV.

ICMP Router Discovery Messages

Gemäß Architekturkonzept aus Kapitel 6 ist vorgesehen, Advertisements mit Hilfe von Multi- oder Broadcasts im Netzwerk zu verbreiten. Die Implementierung der Multicastfunktionalität erwies sich jedoch als sehr komplex. In einem ersten Schritt wurde deshalb diese Funktion mittels *Limited* Broadcasts realisiert. Darüber hinaus wird mit dem Advertisement nur eine einzelne Routeradresse übertragen und deshalb der Wert für den *Preference Level* auf Null gesetzt. In zukünftigen Erweiterungen können hier z. B. zusätzlich auch die IP-Adressen alternativer Kontextrouter übertragen werden. Sowohl im Advertisement- als auch im Solicitation-Paket wurde für das Feld *Code* der Wert 10 gewählt. Dieser Wert ist nach [TyNo08] nicht vergeben, sodass diesbezüglich keine Konflikte zu erwarten sind.

Routingtabellen

Die Servertabelle enthält neben der IP-Adresse, dem Diensttyp und den zu den Diensten gehörenden Kontexttypen auch einen Zeitstempel. Der Server muss sich nach dem Architekturkonzept innerhalb bestimmter Zeitabstände beim Kontextrouter melden, damit dieser Zeitstempel aktualisiert wird. Erfolgt dies nicht, wird der Eintrag gelöscht. Die anderen in Tabelle 6.3 dargestellten Informationen werden durch herkömmliche Routingtabellen des Betriebssystems bereitgestellt.

Um die Funktionen der Weiterleitung und des Rerouting realisieren zu können, wurde in [Wenz07a] auch eine Client-Routingtabelle vorgesehen. Diese enthält den Identifikator, die Sitzungsnummer und den Dienst der Kommunikation. Außerdem werden ein Zeitstempel sowie die IP-Adressen des zuerst ausgewählten Servers und des aktuellen Servers gespeichert. Um die Algorithmen zur Serverauswahl während des Rerouting unterstützen zu können, werden schließlich auch noch die Kontexttypen des ursprünglichen Servers und die vom Client bereitgestellten Kontexttypen mit ihren Prioritäten aufgenommen.

Speicherverwaltung bei Click

Durch die in [Wenz07a] beschriebenen Compiler-Eigenschaften bedingt, können immer nur Paketformate umgesetzt werden, deren Größe ein Vielfaches von 32 Bit besitzen. Daraus folgt, dass alle im Konzept (Kapitel 6) beschriebenen Pakete, die

diese Maßgabe nicht erfüllten, angepasst werden mussten. Im konkreten Fall bedeutet dies, dass die betreffenden Pakete um die entsprechende Anzahl Bytes erweitert und mit Nullen belegt wurden. So musste beispielsweise das AODV-RREQ mit Kontexterweiterung um 2 Bytes ergänzt werden. Einfluss auf die Funktionsweise des kontextsensitiven Routings haben diese Erweiterungen nicht. Auch das dadurch entstehende zusätzliche Datenaufkommen ist nur minimal.

8.2.3.3 Aktueller Stand

Die aktuelle Implementierung des Kontextrouters arbeitet mit Click der Version 1.6.0. Im Folgenden werden noch einige Eigenschaften und Funktionen erläutert, die über den in [Wenz07a] beschriebenen Entwicklungsstand hinausgehen.

Rerouting und NA-Flag

Die Entscheidung darüber, welcher Server bei einem Rerouting bzw. bei der Suche nach einem alternativen Server verwendet wird, hängt von der Serverrangliste ab, die vom Kontextrouter mit Hilfe des entsprechenden Auswahlalgorithmus zusammengestellt wird. Die Zusammenstellung der Liste kann dabei entweder bei jedem Rerouting oder alternativen Anfrage neu – also dynamisch – ermittelt werden. Oder die Auswahl eines geeigneten Servers erfolgt mit einer statischen Rangliste. Diese Liste wird dann bei der ersten Anfrage während der Dienstsuche beim Kontextrouter angelegt und bei jedem weiteren Rerouting verwendet. Beide Implementierungsvarianten sind aktuell realisiert, wobei beim Rerouting mit statischer Rangliste die vom alternativen Server unterstützten Kontexttypen beliebig vom ursprünglichen Server abweichen können. Die Implementierungen haben Vor- und Nachteile bezüglich Reaktionszeit und Aktualität und sollten je nach Anwendungsszenario ausgewählt werden.

Der vom Kontextrouter verwendete Algorithmus für die Auswahl eines für den Nutzer geeigneten Servers kann einfach ausgetauscht und modifiziert werden. Die Übergabe der dazu notwendigen Parameter an den Router erfolgt über Telnet. Darüber hinaus kann natürlich auch der per Default voreingestellte Algorithmus geändert werden. Allerdings muss dann die Software des Routers compiliert werden.

Abmelden von Servern

Im Ad-hoc-Netz kann mit den Funktionen des AODV (z. B. wiederholtes RREQ) festgestellt werden, ob ein Netzknoten noch erreichbar ist. Ein Server ist in einem Infrastrukturnetz in der Regel nur dann nicht erreichbar, wenn er ausgeschaltet wird oder aber irgendein Fehler aufgetreten ist. Ein Fehlerfall sollte dabei im Infrastrukturnetz die Ausnahme sein. Es können nun verschiedene Algorithmen gewählt werden, um die beschriebenen Fälle zu berücksichtigen. Aktuell hat der Server die Möglichkeit, sich vor dem Herunterfahren beim Kontextrouter abzumelden, indem er mit der Registrierungsdatei eine spezielle Nachricht sendet. Der Router kann damit die zugehörigen Routingtabellen aktualisieren. Darüber hinaus muss sich ein Server in bestimmten Abständen beim Kontextrouter melden (siehe auch Abschnitt 8.2.3.2), da sein Eintrag sonst automatisch gelöscht wird. Diese Maßnahmen sind zum stabilen Betrieb des Demonstrators vollkommen ausreichend.

8.2.3.4 Routergraph

Click bietet zur Darstellung der Funktionen des Routers die Möglichkeit, einen Routergraph zu erzeugen. Dieser zeigt die einzelnen Elemente des Routers und deren Verknüpfung untereinander. Aus dem Routergraph der aktuellen Implementierung ergibt sich die in Anhang B befindliche vereinfachte Darstellung. Dort sind aus Gründen der Übersichtlichkeit nur die Elemente und Elementklassen[1] berücksichtigt, die Funktionen zum eigentlichen Routingprozess beitragen. Dagegen wurde auf die Darstellung von Kontroll- und Sicherungsfunktionen (z. B. Kontrolle der *Header Checksum* im IP-Paket), welche allgemein üblich und in der Protokollwelt bekannt sind, verzichtet. Bei Elementen, die mehrfach genutzt werden, ist an den betreffenden Stellen im Graphen die verwendete Instanz angegeben. Auf das ursprüngliche Element wird verwiesen. Das zugehörige aktuelle Skript des Kontextrouters befindet sich in Anhang C. Die im Routergraph enthaltenen Elemente sind in Anhang D aufgelistet. Dort wird auch angegeben, inwieweit diese Elemente aus den Originalquellen von [CMRP07] und [Brae05] übernommen oder modifiziert bzw. im Rahmen dieser Arbeit neu entwickelt wurden. Im Routergraph unterstützt die Netzwerkschnittstelle eth0 ein Ethernet. Alle anderen Netzwerkschnittstellen ethi (mit i=1, 2,. . . , N) bedienen AODV-Ad-hoc-Netze. Gegenwärtig sind davon zwei (N=2) realisiert. Der Routergraph lässt sich in vier funktionale Einheiten aufgliedern, die im Folgenden näher erläutert werden sollen. Alleinstehende schwarz umrandete Elemente dienen als Parameter für andere Elemente. Dort werden Tabellen und Konfigurationsdaten verwaltet und anderen Elementen bereitgestellt.

Eingangseinheit

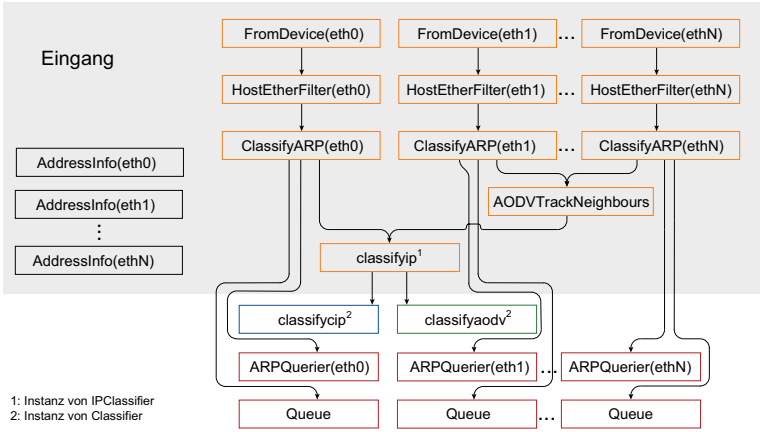


Abbildung 8.3: Die Eingangseinheit des Kontextrouters

Der in Abbildung 8.3 dargestellte Eingangsbereich übernimmt die an den Netzwerkschnittstellen ankommenden Pakete und leitet diese entsprechend ihrer Aufgabe

[1] Elementklassen bestehen i. R. aus mehreren Elementen. Siehe auch Anhang D.

an die einzelnen Funktionseinheiten bzw. Elemente weiter. Das Element *FromDevice(ethi)* liest alle an der zugehörigen Netzwerkschnittstelle i eintreffenden Pakete. Der Parameter *AddressInfo(ethi)* enthält die zu dieser Schnittstelle i zugehörigen Adresseinformationen mit MAC-, IP-Adresse und Subnetzmaske. Mit deren Hilfe und dem Element *HostEtherFilter(ethi)* werden alle Pakete verworfen, die nicht für die Netzwerkschnittstelle i bestimmt sind. Die Elementklasse *ClassifyARP* unterscheidet die Pakete des ARP nach Anfragen, Antworten und restlichem IP-Verkehr. ARP-Antworten, die als Reaktion auf ursprünglich über das Interface gesendete ARP-Anfragen empfangen wurden, werden zur Auswertung an das zugehörige Element *ARPQuerier(ethi)* aus der Ausgangseinheit übergeben. ARP-Anfragen, die das Interface i selbst betreffen, werden beantwortet und zum Versenden an das Element *Queue* in der Ausgangseinheit geleitet. Pakete, die nicht zum ARP gehören, werden im Falle des Ethernets (*ClassifyARP(eth0)*) direkt an das Element *classifyip* gereicht. Bei den zu den AODV-Netzen gehörenden Elementen (Elementklassen *ClassifyARP(eth1)* bis *ClassifyARP(ethN)*) erfolgt die Weiterleitung zuerst an *AODVTrackNeighbours*. Dort werden Hello- und RREP-Pakete des AODV erkannt und die zugehörigen Timer in den AODV-Routingtabellen aktualisiert. *classifyip* unterscheidet schließlich AODV-Pakete und IP-Pakete und leitet diese an die entsprechenden Funktionseinheiten weiter.

AODV-Einheit

In dieser Einheit (Abbildung 8.4) werden alle Funktionen ausgeführt, die das AODV betreffen. *AODVGenerateRREQ* ist verantwortlich für das Initiieren von Anfragen zur Routensuche. Dies erfolgt in Abstimmung mit dem Senden von Hello-Paketen (*AODVHelloGenerator*), um Redundanzen zu vermeiden. Die Pakete werden dann zum Versenden an *outputcl* aus der Ausangseinheit übergeben.

Daneben unterscheidet *classifyaodv* bei den von *classifyip* übergebenen Pakete zwischen fünf Typen – RERR, RREQ, RREP, HELLO und sonstige Pakete. RERR-Pakete werden an *AODVGenerateRERR* übergeben, was die betreffenden Knoten aus den Routingtabellen löscht und die Pakete zur Weiterleitung im Netzwerk an die Ausgangseinheit ausliefert.

Während von *classifyaodv* klassifizierte sonstige Paktete mit *Discard* gelöscht werden, erfolgt dies bei *Hello-Paketen* erst, nachdem über *AODVUpdateNeighbours* die Einträge in der Routingtabelle aktualisiert wurden.

RREQ-Pakete übergibt *classifyaodv* an *AODVUpdateNeighbours*. Dort werden ggf. die Adressinformationen der AODV-Routingtabelle aktualisiert. Das folgende Element *AODVKnownClassifier* überprüft mit Hilfe der Routingtabelle, ob die Route zum gesuchten Knoten bekannt ist. Gilt die Route als unbekannt, muss mit *rreqfwclass* unterschieden werden, ob die Anfragen an ein Ad-hoc-Netz oder ein verbundenes IP-Infrastrukturnetz (hier Ethernet) gerichtet sind. Anfragen für ein Adhoc-Netz werden zur Verbreitung in allen Ad-hoc-Netzen an *outputcl* aus der Ausgangseinheit übergeben. Anfragen zum Ethernet (bzw. IP-Subnetz) muss der Kontextrouter selbst beantworten, da herkömmliche IP-Knoten kein AODV-Protokoll benutzen. Der Router dient damit auch als Bindeglied zwischen heterogenen Netzen. Die Beantwortung übernimmt *GenerateGWRREP*, dessen Ausgabe dann wieder an *outputarpcl* aus der Ausgangseinheit gereicht wird. *outputarpcl* ist deswegen nötig,

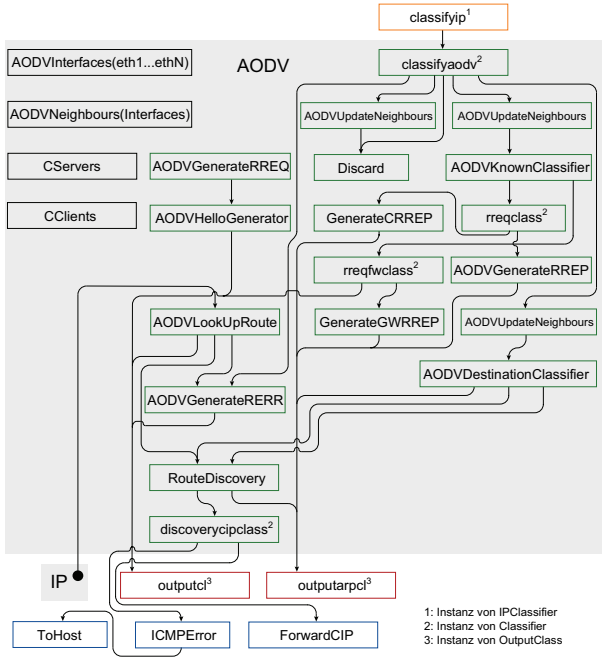


Abbildung 8.4: Die AODV-Einheit des Kontextrouters

weil das neu generierte RREP noch um Header-Informationen der Schicht 2 ergänzt werden müssen.

Sofern *AODVKnownClassifier* feststellt, dass das Ziel des RREQ-Paketes bekannt ist, wird das Paket an das Element *rreqclass* geleitet. Dort wiederum wird zwischen herkömmlichen und für das kontextsensitive Routing erweiterte RREQ-Pakete unterschieden. Herkömmliche RREQ-Pakete erhält *AODVGenerateRREP*, das daraufhin entsprechende RREP-Pakete erzeugt und an *outputarpcl* aus der Ausgangseinheit übergibt. Erweiterte RREQs werden von *GenerateCRREP* verarbeitet. Als Ergebnis wird ein erweitertes RREP erzeugt. Dem Element sind als Parameter *CServers*, *CClients* und *Neighbours* zu übergeben, die den Zugriff auf die verschiedenen Routingtabellen erlauben. *CServers* realisiert dabei auch den Auswahlalgorithmus, mit dem die Rangliste für die Server erzeugt wird. *CClients* verwaltet Informationen zu den anfragenden Clients, die beispielsweise als Basis für die Weiterleitungsfunktion oder das Rerouting dienen. Mit *AODVNeighbours(Interfaces)* wird die AODV-Routingtabelle verwaltet. Dieses Element steht darüber hinaus natürlich auch allen anderen Elementen der AODV-Einheit zur Verfügung, die Zugriff auf die AODV-Routingtabelle erhalten. Der in *Neighbours* übergebene Parameter *AODVInterfaces(eth1...ethN)* enthält dabei Konfigurationsinformationen (Name, IP-Adresse, MAC-Adresse) über die am Router angeschlossenen Netzwerkschnittstellen. Schließlich erfolgt die Übergabe der erweiterten RREPs an *outputarpcl* aus der Ausgangseinheit.

Schließlich werden die von *classifyaodv* als RREP eingeordneten Pakete ebenfalls an *AODVUpdateNeighbours* übergeben. Nach der Aktualisierung der Routingtabelle teilt *AODVDestinationClassifier* die Pakete danach ein, ob sie ursprünglich für den Kontextrouter selbst oder für andere Knoten bestimmt waren. RREP-Pakete, die den Router betreffen, werden an *RouteDiscovery* übergeben, um eine eventuell noch aktive zugehörige Routensuche zu beenden. Außerdem erhalten darüber zwischengespeicherte IP-Pakete die bis dato gesuchte Adresse, sodass diese wiederum über *outputarpcl* aus der Ausgangseinheit weiter verarbeitet werden können. Pakete, die dagegen für andere Knoten bestimmt sind, gehen entweder direkt zu *outputarpcl* aus der Ausgangseinheit oder ebenfalls zur Elementklasse *RouteDiscovery*, sofern keine Route zum Ziel bekannt ist. *RouteDiscovery* übernimmt in diesem Fall die Routensuche und sendet entsprechende RREQs über *outputarpcl* aus der Ausgangseinheit. Für die Generierung der Anfragen wird mit einer Parameterübergabe intern auf die Funktionen von *AODVGenerateRREQ* zurückgegriffen.

Das Element *AODVLookUpRoute* übernimmt die IP-Pakete aus der IP-Einheit, die für die AODV-Netzwerkschnittstellen bestimmt sind, und führt diejenigen mit bekannter Zieladresse *outputcl* in der Ausgangseinheit zu. Pakete, die unbekannte Adressen besitzen, werden an *RouteDiscovery* übergeben, das wiederum einen Suchprozess mittels RREQ initiiert. An ausgefallene oder nicht mehr existente Knoten adressierte Pakete werden an *AODVGenerateRERR* übergeben, welches dann ein RERR generiert und an *outputcl* der Ausgangseinheit leitet.

Die Anzahl der Versuche von *RouteDiscovery*, eine Route zu finden, ist begrenzt. Ist die Suche auch bei der maximal erlaubten Anzahl erfolglos, werden die bis dato zwischengespeicherten Pakete bis auf das letzte verworfen. Dieses letzte IP-Paket wird an *discoverycipclass* übergeben, wo es in die IP-Einheit zurückgegeben wird. Herkömmliche IP-Pakete werden dort an *ICMPError* übergeben, das die in IP spezifizierten Fehlermeldungen generiert und mit *ToHost* an den Kernel des Systems übergibt, der sich um die weitere Behandlung kümmert. Erweiterte IP-Pakete erhält *ForwardCIP*. Ein solches Paket gehört zu einer bis dato aktiven Kommunikation zwischen Client und Server. *ForwardCIP* überprüft die Werte der Paketfelder Identifikator, Session und Diensttyp mit den Daten der entsprechenden Servertabelle. Korrelieren die Werte, muss davon ausgegangen werden, dass der Server nicht mehr erreichbar ist, und er wird aus der Tabelle entfernt. Weitere Erläuterungen zu *ForwardCIP* sind in der nachfolgenden Beschreibung der IP-Einheit zu finden.

IP-Einheit

Die Funktionseinheit für das IP ist in Abbildung 8.5 dargestellt. Die vom Element *classifyip* in der Eingangseinheit selektierten IP-Pakete werden an *classifycip* übergeben. Dort erfolgt eine Selektion nach herkömmlichen und für das kontextsensitive Routing erweiterte IP-Pakete.

Die herkömmlichen IP-Pakete werden an *FilterLocalhost* übergeben. Dieses erkennt wiederum an den Kontextrouter gerichtete Solicitations und initiiert durch Übergabe an *ICMPRouterAdvert(ethi)* ein Advertisement für die Netzwerkschnittstelle i, was zum Versenden direkt an *outputcl* der Ausgangseinheit weitergeleitet wird. Nicht zuzuordnende Solicitations werden verworfen. Daneben sendet *ICMPRouterAdvert(ethi)* in bestimmten Abständen Advertisements über das jeweilige Interface i, sofern während dieser Zeit kein Solicitation empfangen wird. Die Zeitdauer ist

mit Hilfe von *AdvertConfig* konfigurierbar. Erkennt *FilterLocalhost* IP-Pakete, die an eines der Subnetze gerichtet sind, werden diese über *outputcl* der Ausgangseinheit an das entsprechende Interface gereicht. Dies entspricht einem herkömmlichen IP-Routing. Schließlich erkennt *FilterLocalhost* auch solche IP-Pakete, welche direkt an den Kontextrouter gesendet wurden. Davon werden die ICMP-*Echo-Request*-Pakete beantwortet und die verbleibenden Pakete über *ToHost* an den Kernel des Betriebssystems gesendet.

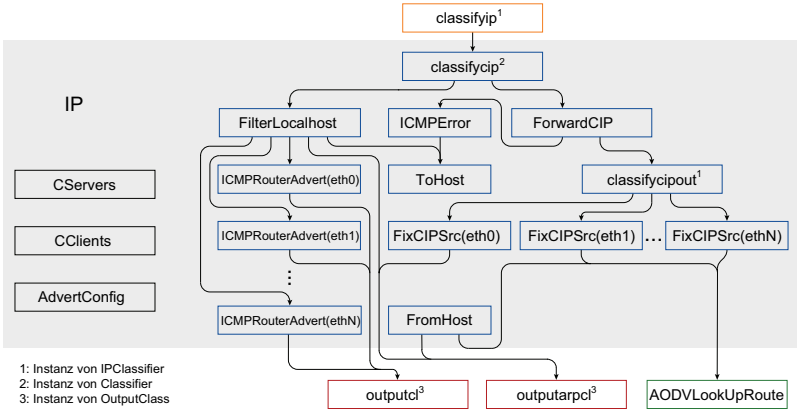


Abbildung 8.5: Die IP-Einheit des Kontextrouters

Die für das kontextsensitive Routing erweiterten IP-Pakete werden von *classifycip* an *ForwardCIP* übergeben. Dieses Element realisiert mit Hilfe von *CServers* und *CClients* die Weiterleitungs- und Reroutingfunktion. Dazu wird der Header der Pakete entsprechend den Vorgaben des Konzeptes modifiziert und an *classifycipout* übergeben. Können erweiterte IP-Pakete anhand der Routingtabellen keiner Kommunikation zugeordnet und ein Rerouting nicht initiiert werden, meldet dies *ICMPError* dem Kernel über *ToHost*.

Das Element *classifycipout* ist verantwortlich dafür, dass die für ein Subnetz i bestimmten Pakete an das zugehörige *FixCIPSrc(ethi)* weitergeführt werden. Dort erhalten die Pakete einen virtuellen Header. Dieser beinhaltet Zusatzinformationen, die auch dann eine Weiterleitung der Pakete an einen Zielknoten erlauben, wenn dieser mehr als ein Hop entfernt ist. Beispielsweise kann *ARPQuerier(ethi)* in der Ausgangseinheit damit Adressinformationen über den unmittelbaren Nachbarknoten, der für die Weiterleitung verantwortlich ist, erhalten.

Wenn das IP-Paket für das Ethernet bestimmt war, wird es direkt an *outputarpcl* aus der Ausgangseinheit übergeben. Pakete für die Ad-hoc-Netzwerkschnittstellen werden an *AODVLookUpRoute* der AODV-Einheit zur Weiterverarbeitung übergeben. Deren Funktion wurde bereits oben erläutert. Ebenso wird mit IP-Paketen verfahren, die vom Kernel des Systems über *FromHost* übergeben werden.

Ausgangseinheit

Für die Übergabe der Pakete an die einzelnen Netzwerkschnittstellen ist die in Abbildung 8.6 gezeigte Ausgangseinheit verantwortlich. *outputcl* weist den zu versen-

denden Paketen die Warteschlange *Queue* der zugehörigen Netzwerkschnittstelle i zu. Pakete, die an *outputarpcl* übergeben werden, besitzen noch keinen Schicht-2-Rahmen und damit auch keine MAC-Adresse. Ist die MAC-Adresse des Ziels zudem nicht bekannt, wird eine ARP-Anfrage in das Netzwerk gesendet und zusammen mit *ClassifyARP(ethi)* aus der Eingangseinheit ausgewertet. Bleibt diese Anfrage erfolglos, werden die zugehörigen Pakete verworfen. *ToDevice(ethi)* stellt schließlich die Schnittstelle zur eigentlichen Netzwerkkarte des Netzwerks i dar.

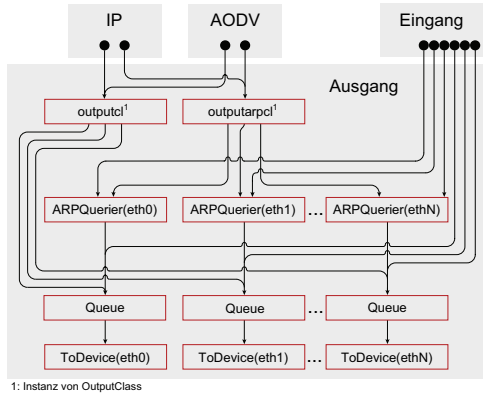


Abbildung 8.6: Die Ausgangseinheit des Kontextrouters

8.3 Client

Bei der Realisierung des Clients wurden die in Abschnitt 6.3.4 beschriebenen Funktionen umgesetzt. Dabei war wie beim Kontextrouter zu beachten, dass die Anforderungen an das erweiterte AODV im Infrastrukturnetz abweichen. Bestimmte Funktionen (z. B. Hello-Pakete) sind dort überflüssig, weshalb darauf verzichtet wird. Die den Client betreffenden Funktionen des kontextsensitiven Routings müssen dagegen vollständig unterstützt werden, um die korrekte Funktion des Architekturkonzeptes nachweisen zu können. Dazu gehören entsprechend dem Konzept aus Kapitel 6:

- Generieren von Solicitations
- Auswerten von Advertisements
- Generieren erweiterter AODV-RREQ
- Auswerten erweiterter AODV-RREP
- Unterstützung der erweiterten IP-Pakete
- ggf. Rerouting-Option
- ggf. Unterstützung einer *Black List*
- ggf. manuelle Eingabe von Kontextrouteradressen

8.3.1 Aktueller Stand

Prinzipiell kann die Software Click auch zur Programmierung der eigentlichen Kommunikationspartner Client und Server verwendet werden. Diesbezüglich wurden in [Wenz07a] verschiedene Elemente programmiert, die es ermöglichten den Kontextrouter zu testen. Mit den zugehörigen Skripten können folgende Funktionen des Clients realisiert werden:

- Generierung einzelner Solicitations
- Generierung einzelner erweiterter RREQs
- Generierung erweiterter IP-Pakete

Bei den Skripten für die ersten beiden Funktionen wird jeweils ein einzelnes entsprechend den Anforderungen präpariertes Paket gesendet, während es sich bei der Generierung erweiterter IP-Pakete um einen Datenstrom aus modifizierten ICMP-Ping-Paketen handelt. Diese Funktionen reichen vollkommen zum Verifizieren des Architekturkonzeptes aus.

Software	AODV-UU	AODV-UIUC	UoB-JAdhoc	UoBWinAODV	Kernel-AODV
RFC 3561	ja	nein	ja	ja	ja
Modifikation Routing	User-Space	User-Space	User-Space	User-Space	Kernel
Plattform	Linux	Linux	unabhängig (Java)	Windows XP	Linux
Kernel	2.4 & 2.6	2.4	–	NT 5.1	2.4
letzte Version	0.9.5	0.5	0.21	0.15	2.2.2
vom	23.07.2007	30.07.2004[2]	12.04.2005	21.03.2005	15.04.2004

Tabelle 8.2: AODV-Implementierungen

Darüber hinaus wurde bereits untersucht, wie eine Implementierung für ein Produktionsnetz realisiert werden kann. Ein Ergebnis aus [Renh07a] ist dabei, dass für die Umsetzung des Clients zwar ebenfalls Click nutzbar wäre, aber auch geeignetere Lösungen existieren. Das liegt vor allem daran, dass Click entwickelt wurde, um Router möglichst komfortabel zu realisieren. Clients sind aber Netzknoten, die selbst im Adhoc-Netz auf verschiedene Funktionen eines Kontextrouters verzichten können, und somit andere Anforderungen an eine Auswahl geeigneter Software stellen. Hier sind bereits existierende Implementierungen, die AODV unterstützen, besser als Basis geeignet, da sich kleinere Modifikationen effizienter als bei Click durchführen lassen. Einen Überblick aktuell verfügbarer Software zeigt Tabelle 8.2. Es handelt sich hierbei ausschließlich um Entwicklungen von Universitäten und Forschungsinstituten. Bis auf die Software AODV-UIUC [KaTG03] unterstützen alle Implementierungen die Funktionen für AODV nach dem RFC 3561. Damit könnten zwar UoB-JAdhoc [UoBJ07], UoBWinAODV [UoBW07] und Kernel-AODV [Klei07] als Basis für den

[2] bezieht sich auf die letzte Änderung der zugehörigen Bibliothek

Client dienen. Tatsächlich werden diese Versionen aber seit längerer Zeit nicht mehr gepflegt. Hinzu kommt, dass der Routingalgorithmus beim Kernel-AODV nur im Kernel modifiziert werden kann. Jede Änderung müsste durch neues Compilieren des Kernels in das System übernommen werden. Im Gegensatz dazu unterstützt dies das auf der Programmiersprache C basierende AODV-UU [Nord07] im so genannten *User-Space*, d. h. im Anwendungsbereich. Außerdem wird diese Software aktuell weiter entwickelt bzw. gepflegt und arbeitet auch in Linuxsystemen, die auf dem Kernel der Version 2.6 basieren. Damit ist diese AODV-Implementierung am besten als Basis für die Realisierung des Clients geeignet.

Die Entscheidung darüber, AODV-UU für die Realisierung des Clients zu nutzen, hatte auch unmittelbar Einfluss auf die Festlegung des Wertes für das *Type*-Feld in den Erweiterungen von RREQ- und RREP-Paketen (siehe Abbildungen 6.16 bzw. 6.17 in Abschnitt 6.3.4). Da beim AODV-UU schon verschiedene Werte für dieses Feld vergeben sind und mit Rücksicht auf den RFC 3561 [BRPD03], gilt für das kontextsensitive Routing folgende Vereinbarung: 16 für RREQ und 17 für RREP.

8.3.2 Zukünftige Arbeiten

Wie bereits beschrieben, existieren erste prototypische Implementierungen, die die Funktionen des Clients realisieren. Daneben wird gegenwärtig an einer integrierten Lösung sämtlicher Clientfunktionen auf Basis von AODV-UU gearbeitet. Erste Ergebnisse wurden bei [Renh07a] vorgestellt. Danach erfolgt die Übergabe der Anfrage nach kontextsensitiven Diensten vorerst textbasiert. Ziel ist die vollständige Umsetzung der zu Beginn von Abschnitt 8.3 dargestellten Funktionen innerhalb einer Implementierung, die als Basis für einen zukünftigen Einsatz in einem Produktionsnetz dient.

Eine besondere Herausforderung wird bei zukünftigen Forschungs- und Entwicklungsarbeiten die Schnittstelle zur Anwendungsschicht darstellen. Es muss hierfür eine Lösung gefunden werden, die es den Anwendungen auf einfache Weise erlaubt, den gesuchten Dienst und die aktuell zur Verfügung stehenden Kontexttypen zu übergeben. Die vom Client unterstützten Kontexttypen können dabei einerseits direkt von der Anwendung übernommen werden. Andererseits könnte dies auch vom System und somit unabhängig von der Anwendung erfolgen. Damit würden bei einer Dienstsuche von Anfang an alle zur Verfügung stehenden Kontexttypen berücksichtigt. Die Anwendung muss nur erkennen können, ob der Client die obligatorischen Kontexttypen erbringen kann. Ist das nicht der Fall, wird dies dem Nutzer mit einer Fehlermeldung angezeigt. Werden diese Kontexttypen unterstützt, erfolgt die weitere Dienstsuche unabhängig von der Anwendung. Der Client überträgt dazu alle verfügbaren (obligatorischen und optionalen) Kontexttypen. Die sich daraus ergebenden Kontextinformationen werden dann vielleicht nicht unmittelbar für die Anwendung benötigt, können aber zur verbesserten Diensterbringung beitragen. Da sich die Dienste beispielsweise immer weiter entwickeln, könnten diese durchaus zukünftig mehr Kontexttypen auswerten als ursprünglich einmal vorgesehen war.

Darüber hinaus muss noch erforscht werden, welche Ansätze zur Realisierung dieser Schnittstelle zu bevorzugen sind. Eine Möglichkeit wäre beispielsweise, eine einzelne Schnittstelle zu definieren, über die die entsprechenden Anwendungen zugreifen können. Eine andere Möglichkeit besteht darin, ein API zur Verfügung zu stellen, wodurch den Programmierern von Anwendungen eine flexible Zugangsmöglichkeit

geboten würde. Allerdings ist diese Lösung auch komplexer. Letztlich müssen zukünftige Forschungs- und Entwicklungsarbeiten entscheiden, wie eine entsprechende Schnittstelle gestaltet werden kann.

8.4 Server

Im Gegensatz zum Kontextrouter und zum Client benötigt der Server keine Funktionen, die eine Dienstsuche unterstützen. Auch wird nur dann das AODV-Protokoll benötigt, wenn sich der Server in einem entsprechenden Ad-hoc-Netz befindet. Für diesen Fall kann beispielsweise auf die in Abschnitt 8.3.1 favorisierte Implementierung AODV-UU zurückgegriffen werden. Für den Server müssen mit den Anforderungen aus Kapitel 6 somit folgende Funktionen realisiert werden:

- Generieren von Solicitations
- Auswerten von Advertisements
- Registrierung beim Kontextrouter
- Unterstützung der erweiterten IP-Pakete
- ggf. manuelle Eingabe von Kontextrouteradressen

8.4.1 Aktueller Stand

Für den Test des Kontextrouters wurde ebenfalls in [Wenz07a] ein Skript für Click entwickelt. Dieses Skript wertet empfangene für das kontextsensitive Routing modifizierte ICMP-Ping-Pakete aus und antwortet entsprechend. Realisiert wird die Funktionalität mit Hilfe eines zusätzlichen Elementes.

Eine weitere Implementierung basiert auf der Programmiersprache C und wurde im Rahmen von [Domb07] entwickelt. Damit können die Solicitation-/Advertisement-Funktionen ausgeführt werden. Diesbezüglich musste jedoch noch eine Anpassung auf *Limited*-Broadcast-Adressen erfolgen. Darüber hinaus können mit der Implementierung auch Dienste beim Kontextrouter registriert werden. Für die Übergabe der Dienste und unterstützten Kontexttypen steht eine Web-Schnittstelle bzw. der direkte Zugriff auf die entsprechenden Dateien zur Verfügung. Die erzeugte Liste mit den vom Server unterstützten Diensten und Kontexttypen wird mit Hilfe der in Abschnitt 8.2.3.1 bereits beschriebenen Prozedur via SSH an den Kontextrouter übertragen.

Beide vorgestellten Implementierungen ergänzen sich und unterstützen somit eine vollständige Verifikation des Architekturkonzeptes und insbesondere des Kontextrouters.

8.4.2 Zukünftige Arbeiten

Im Zuge weiterer Entwicklungsarbeiten sollte die Software für den Server aus [Domb07] um die Möglichkeit der Kommunikation mit erweiterten IP-Paketen ergänzt werden. In einem weiteren Schritt und zu Testzwecken können dann fiktive Dienste generiert werden, auf die der Server bei entsprechenden Anfragen reagiert (z. B. Dateitransfer). Zum Nachweis der Funktionsfähigkeit des kontextsensitiven Routings wird dies zwar nicht benötigt. Allerdings muss zukünftig auch für den Server ein Gesamtkonzept bezüglich eines praktischen Einsatz bereitgestellt werden.

8.5 Kapitelzusammenfassung

Im vorliegenden Kapitel wurde beschrieben, wie die für das Konzept aus Kapitel 6 benötigten Funktionseinheiten bzw. Netzknoten praktisch realisiert wurden. Aktuell existieren Implementierungen für den Kontextrouter, den Client und den kontextsensitiven Server. Es wurde gezeigt, dass für die Realisierung des Kontextrouters die Nutzung des Softwarewerkzeugs Click am sinnvollsten ist. Anhand des beschriebenen Routergraphen konnte die Funktionsweise des Kontextrouters erklärt werden. Alle Anforderungen aus dem Konzept wurden erfüllt.

Des Weiteren stehen für den Client Skripte in Click bereit, mit denen die Architektur für das kontextsensitive Routing vollständig verifiziert werden kann. Aktuell wird darüber hinaus eine Software basierend auf AODV-UU entwickelt, die das Spektrum der Funktionen des Clients innerhalb einer einzelnen Implementierung realisiert. Für den kontextsensitiven Server steht bereits eine Implementierung zur Verfügung, die in Verbindung mit einem weiteren Click-Skript ebenfalls sämtliche in dem Konzept geforderten Funktionen realisiert.

Der Schwerpunkt bei der Umsetzung des Architekturkonzeptes nach Kapitel 6 lag bisher auf der Entwicklung des Kontextrouters. Client und Server dienen aktuell lediglich als Testinstrumente und sollen zukünftig weiter ausgebaut werden. Dazu wurden erste Ansätze und Vorschläge beschrieben. Besondere Beachtung sollte beim Client die Verbindung zwischen Routingsoftware und Anwendung finden.

Bei allen Implementierungen wurde und wird darauf geachtet, dass eine Kompatibilität zu herkömmlichen IP-Netzwerken besteht. Damit ist eine einfache IP-Kommunikation genauso möglich wie eine einfache Dienstanfrage auch. Außerdem werden sowohl Schnittstellen zum Ad-hoc-Netz als auch zum Infrastrukturnetz unterstützt. Ob die Funktionen nach den Vorgaben des Konzeptes realisiert wurden und was das kontextsensitive Routing tatsächlich leistet, können nur entsprechende Tests und Messungen nachweisen. Diesem Thema widmet sich das folgende Kapitel.

9. Verifikation der praktischen Realisierung

Das folgende Kapitel weist die erfolgreiche Umsetzung des Architekturkonzeptes nach. Nach einer Einführung in die Thematik werden dazu die Messumgebung vorgestellt und die durchgeführten Tests zu den im Konzept geforderten Funktionen sowie die zugehörigen Ergebnisse erläutert. Darüber hinaus wird ein erster Test hinsichtlich der Leistungsfähigkeit durchgeführt.

9.1 Einführung

Da zu Beginn keinerlei Erfahrungen auf dem Gebiet der Realisierung von Ad-hoc-Netzen existierten, erfolgten diesbezüglich im Vorfeld der Umsetzung des Architekturkonzeptes mehrere Untersuchungen und Testaufbauten. Stellvertretend sollen hierfür [Kram05, Marr06, Nowa06, Psch06] genannt werden. Dort wurden zum einen verschiedene AODV- und OLSR-Implementierungen auf ihre Funktion hin untersucht. Dabei wurde auf unterschiedliche Hardware (PCs, Laptops, PDAs) und Betriebssysteme (z. B. Windows 2000, Windows XP, diverse Linux-Distributionen) zurückgegriffen. Zum anderen erfolgten bereits Untersuchungen hinsichtlich des Verhaltens in heterogenen Netzwerkumgebungen sowie der Unterstützung multimedialer und kontextsensitiver Anwendungen. Zwar wurden die zuletzt genannten Untersuchungen zu den Eigenschaften von Ad-hoc-Netzen mit dem proaktiven OLSR durchgeführt. Trotzdem lassen sich die Ergebnisse auf AODV-Netzwerke übertragen, sofern sich diese auf Eigenschaften beziehen, die über die eigentliche Routensuche hinaus gehen. So kann beispielsweise ein Betrieb in heterogenen Netzwerkumgebungen nur über Hilfsmittel (z. B. Gateways) erfolgen. Auch spielt die Verwendung eines Ad-hoc-Netzes für das Erbringen kontextsensitiver Dienste nur dann eine Rolle, wenn diese echtzeitkritisch und die Netzknoten sehr mobil sind. Problematisch ist hier, wie bereits in Abschnitt 3.2 beschrieben, die Umschaltzeit für einen Handover. Dies ist jedoch lediglich für die Realisierung des jeweiligen Dienstes von Bedeutung und liegt nicht im Fokus dieser Arbeit. Außerdem hat das kontextsensitive Routing keinen zusätzlichen Einfluss auf diese Eigenschaft, da es die Wegefindung des herkömmlichen AODV nicht verändert, sondern lediglich dessen Funktionen erweitert.

Darüber hinaus konnten auch bezüglich der Messszenarien Erfahrungen gesammelt werden. Als ein häufig auftretendes Problem stellte sich dabei die Realisierung von definierten Handovern heraus. Die betreffenden Knoten mussten dazu außerhalb der gegenseitigen Reichweite bewegt werden. Störungen durch zusätzliche „fremde“ WLANs, Gebäudestrukturen, Personen im Messumfeld usw. äußerten sich dabei durch unterschiedliche Reichweiten und Umschaltzeiten beim Handover. Als hilfreich erweist sich hier der *Wireless Network Topology Emulator* (WNTE [wnte03]), mit dem auf einfache Art und Weise ein Handover erzwungen werden kann. Mit dem in diesem Softwarepaket enthaltenen Werkzeug iptmac können auf einem Knoten die MAC-Adressen von bestimmten benachbarten Knoten gesperrt werden, sodass diese sich über andere Knoten ein Weg zu ihrem Kommunikationsziel suchen müssen. Dadurch können theoretisch auch mit statischen Knoten beliebige Ad-hoc-Netz-Topologien konfiguriert werden.

Um die Funktionen des Konzeptes jedoch praktisch verifizieren zu können, musste ein Demonstrator bzw. ein Demonstratornetzwerk aufgebaut werden. Dazu waren alle Funktionseinheiten des Architekturkonzeptes umzusetzen. Neben dem Kontextrouter beinhaltet dies wenigstens einen Server und einen Client. Für Tests, die das Rerouting oder die Änderung der Topologie im Ad-hoc-Netz betreffen, sind entsprechend mehr Knoten notwendig. Den Aufbau des aktuellen Demonstratornetzwerkes beschreibt der folgende Abschnitt.

9.2 Messumgebung

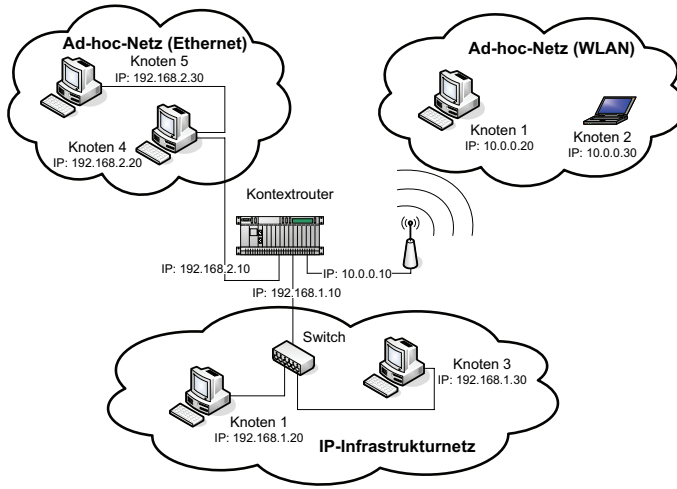


Abbildung 9.1: Demonstrator

In Abbildung 9.1 ist der Demonstrator in seiner aktuellen Konfiguration und mit den derzeit verfügbaren Knoten dargestellt. Hierbei handelt es sich um ein unabhängiges und eigenständiges Netzwerk. Herzstück bildet dabei der als Kontextrouter arbeitende PC. Dessen Realisierung mit Hilfe von Click wurde bereits in Kapitel 8 beschrieben. Der Kontextrouter verbindet mehrere Subnetze miteinander. Die IP-Adressen

der dort implementierten Knoten sind in der Abbildung angegeben. Zu beachten ist, dass die IP-Adressen 192.168.1.20 und 10.0.0.20 jeweils demselben physischen Knoten (mit zwei Schnittstellen zu den entsprechenden Netzwerken) in der Messumgebung zugeordnet wurden.

In der aktuellen Konfiguration werden an den Schnittstellen des Kontextrouters drei verschiedene Netzwerktypen realisiert. Neben einem IP-Infrastrukturnetz und einem Ad-hoc-Netz, das entsprechend den Erkenntnissen aus Abschnitt 2.3 mit WLAN realisiert wurde, ist ein zweites Infrastrukturnetz implementiert, das allerdings ebenfalls Ad-hoc-Netz-Funktionalität besitzt. Der Einsatz dieses Ethernets erfolgte bereits in [Wenz07a] unter Verwendung von Click. Der weiterleitende Knoten 4 ist dazu mit zwei Ethernetkarten ausgestattet, die beide eine logische Schnittstelle bilden. Über diese Schnittstelle kann dann mit dem Kontextrouter und mit dem benachbarten Knoten kommuniziert werden. Allerdings stellt ein solches Netz lediglich eine Hilfsmaßnahme dar. Der Grund für eine solche Realisierung bestand zum einen darin, dass zu Beginn des Demonstratoraufbaus nur eine begrenzte Anzahl von WLAN-Karten bzw. Endgeräten zur Verfügung stand. Zum anderen bietet die Hardware des Kontextrouters bessere Möglichkeiten, mehrere Ethernetschnittstellen gleichzeitig bereitzustellen als dies mit WLAN-Karten möglich wäre. Mit Hilfe von Ethernet-Netzwerkkarten können somit auch zukünftig auf einfache Art und Weise mehrere Ad-hoc-Netze parallel emuliert werden. Für Funktionstests ist diese Variante zwar nutzbar, bei Performancetests sollte darauf jedoch verzichtet werden, weil sich die Eigenschaften (z. B. unterstüzte Bitraten, Medienzugriff usw.) bei WLAN und Ethernet doch sehr unterscheiden.

Der in Abbildung 9.1 dargestellte Demonstrator lässt sich beliebig erweitern. Durch Modifikation des in Abschnitt 8.2.3.4 vorgestellten Routergraphes lassen sich beim Kontextrouter weitere Schnittstellen hinzufügen. Die Zahl der Knoten in den einzelnen Subnetzen kann ebenfalls beliebig verändert werden. Alle derzeit involvierten Knoten arbeiten mit einem Linux-Betriebssystem. Die dabei verwendete Distribution Gentoo nutzt den Linux-Kernel Version 2.6. Die einzelnen Knoten können sowohl als Server als auch als Clients verwendet werden. Die dazu benötigte Software wurde ebenfalls bereits in Kapitel 8 beschrieben. Zum besseren Verständnis der Testergebnisse werden in den folgenden Abbildungen zu den Versuchsanordnungen für jeden Knoten der Name und für die zugehörigen Netzwerkschittstellen die IP- sowie MAC-Adressen angegeben.

An die Hardware der Knoten wurden keine besonderen Anforderungen gestellt. Da Linux sehr ressourcenschonend arbeitet, reichen die Minimalanforderungen aus. Somit können die Knoten innerhalb des Demonstrators auch untereinander leicht ausgetauscht werden. Als Kontextrouter dient beispielsweise ein PC mit einem Pentium-III-Prozessor (800 MHz) und 256 MByte Hauptspeicher. Dort haben sich weder die grafische Oberfläche von Linux noch die Erweiterung um Netzwerkschnittstellen auf dessen Leistungsfähigkeit ausgewirkt. Sollten jedoch größere Netzwerke realisiert oder ressourcenintensive Performancetests durchgeführt werden, kann dies durchaus Auswirkungen auf die Messergebnisse haben und ist entsprechend zu berücksichtigen. Für Funktionstests spielt dies jedoch keine Rolle. Alle bei den folgenden Tests beteiligten Rechner sind mit ihren Leistungsdaten in Anhang E aufgeführt.

In der aktuellen Entwicklungsstufe ist es möglich, dem Kontextrouter den Algorithmus für die Auswahl eines passenden Servers via Option zu übergeben. Darüber

hinaus kann eine Vielzahl von Parametern über eine Konfigurationsdatei modifiziert werden. Ein Auszug dieser Parameter befindet sich im Anhang A. Der Kontextrouter wird dadurch zu einem flexiblen Testwerkzeug und bildet somit auch eine solide Basis für zukünftige Forschungen. Im Folgenden werden die einzelnen Funktionstests und deren Ergebnisse erläutert. Die Auswertung der Kommunikationsprozesse erfolgte mit Hilfe des Werkzeugs Wireshark und ggf. den Debug-Ausgaben der für die Kommunikation eingesetzten Software. Wireshark eignet sich hierbei besonders gut für die Analyse von Frames und Paketen. Die Analysemöglichkeiten dieser Software werden in Abschnitt 9.3.1.2 näher erläutert.

9.3 Funktionstest

Die nachfolgend beschriebenen Tests dienten zum Nachweis der relevanten Funktionen des Architekturkonzeptes. Die Reihenfolge richtete sich dabei nach den kausalen Abhängigkeiten. Damit konnten bereits getestete Funktionen jeweils als Basis nachfolgender Funktionstests dienen. Ausgangspunkt war die Initialisierung des Kontextrouters und der Netzknoten. Daran schloss sich die Dienstregistrierung durch einen Server, die Dienstanfrage durch einen Client und die Weiterleitung der Daten bei der Dienstkommunikation zwischen diesen beiden Knoten an. Darüber hinaus erfolgten sowohl Tests zur Weiterleitung unmodifizierter IP-Pakete als auch zum Rerouting und zum Verhalten in einer konventionellen AODV-Umgebung.

9.3.1 Advertisements und Solicitation

In diesem Test wurde überprüft, ob die Funktionen für Advertisements und Solicitations entsprechend den Abschnitten 6.3.2.1 und 6.3.2.2 ausgeführt werden. Dazu gehören zum einen das selbständige Versenden von Advertisements durch den Kontextrouter nach dessen Initialisierung und während des laufenden Betriebs. Zum anderen muss der Kontextrouter auf Solicitations anderer Netzknoten korrekt reagieren.

9.3.1.1 Durchführung

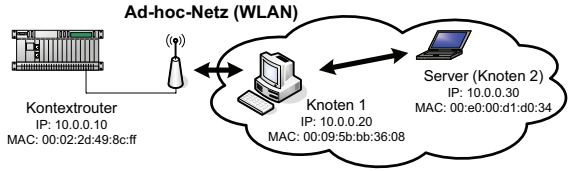


Abbildung 9.2: Anordnung für Advertisement/Solicitation-Test

Die Durchführung dieses Tests erfolgte im Ad-hoc-Netz, sodass auch verifiziert werden konnte, ob eine Multihop-Kommunikation möglich ist. Abbildung 9.2 zeigt die dabei beteiligten Knoten. Die aktuelle Version von AODV-UU unterstützt kein Broadcast, deshalb wird auf Knoten 1 ein spezielles Click-Skript benutzt, das die Weiterleitung von Advertisements/Solicitations mit Hilfe von *Limited* Broadcasts realisierte. Auf Knoten 2 wurde der in Abschnitt 8.4.1 beschriebene Server von [Domb07] zusammen mit AODV-UU eingesetzt. Da sich in diesem Test alle Knoten und der Kontextrouter unmittelbar in gegenseitiger Reichweite befanden, wurde

der Knoten 2 mit Hilfe von iptmac so konfiguriert, dass dieser die Pakete des Kontextrouters ignoriert. Da der Kontextrouter selbst im *Userlevel*-Modus lief, iptmac jedoch nur auf Kernelebene arbeitet, musste die Konfiguration dort im Routerskript erfolgen. Dieses wurde um einen entsprechenden Eintrag (siehe Anhang C) ergänzt, sodass der Kontextrouter nun alle von Knoten 2 empfangenen Pakete ignoriert hat.

9.3.1.2 Auswertung

Wie bereits erwähnt, wurde zur Auswertung der in diesem Kapitel durchgeführten Tests das Programm Wireshark verwendet. Zum besseren Verständnis der dabei aufgenommenen Analysedaten erfolgt mit Hilfe der Abbildung 9.3 eine kurze Einführung in das Programm. Neben dem allgemein üblichen Menübereich[1], verfügt das Programm über einen Filter, der das gezielte Anzeigen von Paketen (z. B. nach Protokollzugehörigkeit) ermöglicht. Wireshark verfügt über drei Ausgabefenster, um die an der betreffenden Netzwerkschnittstelle erkannten Pakete (hier als Frames bezeichnet) auszuwerten. Im oberen Ausgabefenster werden die Frames angezeigt. Dort wird unter anderem auch das zu den Daten im jeweiligen Frame gehörende Protokoll der höchsten Protokollschicht angegeben. Das mittlere Ausgabefenster zeigt die analysierten Daten eines Frames an und ordnet diese den einzelnen Protokollschichten zu. Des Weiteren wird hier die Bedeutung der zu den einzelnen Protokollen gehörenden Daten ausgegeben, sofern diese bekannt bzw. spezifiziert sind. Das untere Ausgabefenster des Programmes ist vor allem für den Bereich der Protokollentwicklung interessant. Dort wird der Inhalt des jeweiligen Frames hexadezimal ausgegeben. Damit ist es möglich, auch neue und noch nicht standardisierte Werte eines Protokolls zu überprüfen. Zusätzlich ist der Offset der Daten (links im Ausgabefenster) und, falls dies möglich ist, deren ASCII-Kodierung (rechts) zu sehen. Die beiden unteren Ausgabefenster von Wireshark lassen sich für ein einzelnes Frame auch zusammen in einem separaten Fenster anzeigen.

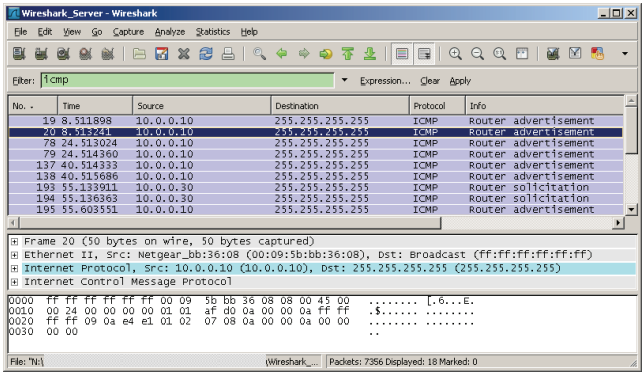


Abbildung 9.3: Wireshark auf Knoten 2: Solicitation und Advertisement

Die Ergebnisse des durchgeführten Tests zur Überprüfung von Solicitations und Advertisements sind wie folgt zu interpretieren. Beim Start des Kontextrouters sendete

[1]Im Folgenden wird bei Screenshots von Wireshark auf die Darstellung von Menü sowie unterem Rand verzichtet und jeweils nur noch ein Auszug mit den für die Testauswertung relevanten Informationen gezeigt.

dieser entsprechend dem RFC 1256 [Deer91] in alle Netzwerke zwei Advertisement-Pakete im Abstand von 16 s. Die weiteren Abstände wurden dann zufällig aus einem Intervall zwischen 450 s und 600 s gewählt. Der Kontextrouter bestätigte dies durch folgende Ausgabe:

```
ICMPAdvert: next advertisement in 16000ms at 192.168.1.10
ICMPAdvert: next advertisement in 16000ms at 192.168.2.10
ICMPAdvert: next advertisement in 16000ms at 10.0.0.10
...
ICMPAdvert: next advertisement in 16000ms at 192.168.1.10
ICMPAdvert: next advertisement in 16000ms at 192.168.2.10
ICMPAdvert: next advertisement in 16000ms at 10.0.0.10
ICMPAdvert: next advertisement in 578939ms at 192.168.1.10
ICMPAdvert: next advertisement in 578939ms at 192.168.2.10
ICMPAdvert: next advertisement in 578939ms at 10.0.0.10
IMCPAdvert: solicit received from 10.0.0.30
ICMPAdvert: next advertisement in 485003ms at 10.0.0.10
...
ICMPAdvert: next advertisement in 511212ms at 10.0.0.10
ICMPAdvert: next advertisement in 579469ms at 192.168.1.10
ICMPAdvert: next advertisement in 579469ms at 192.168.2.10
```

Nachdem der Kontextrouter von Knoten 2 ein Solicitation empfangen hatte, wurde in das betreffende Netzwerk ein Advertisement gesendet. An dieser Stelle ist anhand der Ausgabe des Kontextrouters zu sehen, dass die Behandlung der Advertisements für die einzelnen Netzwerke tatsächlich unabhängig voneinander war. Die Zeitdauer zwischen zwei Advertisements unterscheidet sich hier für das Netzwerk 10.0.0.0 von den anderen beiden Netzwerken. Dass das Solicitation-Paket gesendet und darauf ein Advertisement empfangen wurde, illustriert auch die Ausgabe von Wireshark auf dem Knoten 2 in Abbildung 9.3. Auffällig ist dort, dass alle Pakete anscheinend dupliziert wurden. Anhand der MAC-Adressen ließ sich aber nachvollziehen, dass es sich eigentlich nur um drei empfangene Advertisements handelte. Nach diesen wurde beim Knoten 2 ein Solicitation initiiert, worauf der Kontextrouter mit einem weiteren Advertisement reagierte. Nähere Untersuchungen ergaben dann, dass Wireshark alle an einer Schnittstelle ankommenden Pakete unabhängig ihrer weiteren Verarbeitung anzeigt. In Abbildung 9.4 werden die scheinbar duplizierten Solicitation-Pakete miteinander verglichen. Dort stimmen die IP-Adressen (siehe Zeile: `Internet Protocol`) von Quelle und Ziel überein. Anhand der MAC-Adressen (siehe Zeile: `Ethernet II`) wird jedoch ersichtlich, dass hier sowohl das Solicitation des Knotens 2 (MAC 00:E0:00:D1:D0:34) als auch das weitergeleitete Paket von Knoten 1 (MAC 00:09:5B:BB:36:08) angezeigt wird. Auch die Zeitstempel bestätigen dies. Es wird immer das Paket der ursprünglichen Quelle zuerst und danach das vom weiterleitenden Knoten ausgegeben. Natürlich wird dabei vom Kontextrouter nur das weitergeleitete Paket ausgewertet. Deshalb antwortet dieser auch mit nur einem Advertisement.

Entsprechend den Vorschlägen aus den Abschnitten 6.3.2.1 und 6.3.2.2 sowie den Ausführungen zur praktischen Umsetzung in Abschnitt 8.2.3.2 wurde das Feld *Code* verwendet, um die Solicitations/Advertisements zum Suchen/Finden eines Kontextrouters zu kennzeichnen. Deshalb wird in Abbildung 9.4 für das Feld *Code* der Wert 10 angezeigt.

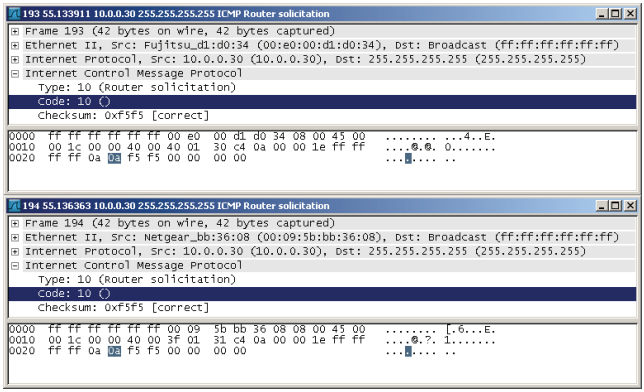


Abbildung 9.4: Vergleich der Solicitation-Pakete

Insgesamt konnte mit diesem Test nachgewiesen werden, dass sich der Kontextrouter beim Versenden von Advertisements entsprechend den Parametern der Spezifikation RFC 1256 verhält. Ebenso werden Solicitation-Pakete beantwortet. Die Konfiguration der Knoten zeigt, dass es keine Interoperabilitätsprobleme zwischen der AODV-Implementierung mittels Click (Kontextrouter, Knoten 1) und AODV-UU (Knoten 2) gibt. Die Tests sind genauso erfolgreich mit *Directed* Broadcasts absolviert worden. Diese Adressierung entspricht zwar nicht dem RFC 1256, ist aber einer Multicast-Adresse für ein Subnetz ähnlich.

Leider hatte sich gezeigt, dass die Filterung von MAC-Paketen zu unvorhergesehenen Effekten führt. Die Analyse der Kommunikationsdaten wird beispielsweise dadurch unübersichtlich, weil alle an einer Schnittstelle ankommenden incl. der eigentlich gefilterten Pakete von Wireshark angezeigt werden. Das betrifft auch andere Sniffing-Werkzeuge wie z. B. tcpdump. Weitere Tests zeigten auch Probleme bei der Kommunikation mit höheren Protokollen, sofern darüber direkt auf den Kontextrouter zugegriffen wurde. Es ist aber davon auszugehen, dass diese sich lösen, sobald der Kontextrouter im *Kernel*-Modus betrieben wird. Dann kann WNTE tatsächlich als sehr effizientes Hilfsmittel zur Topologiesteuerung dienen. Um aktuell die Ergebnisse jedoch nicht weiter zu beeinflussen, wurde im Folgenden auf die Möglichkeit der MAC-Filterung verzichtet und die Kommunikation über Knotengrenzen hinweg durch räumlichen Abstand realisiert.

9.3.2 Dienstregistrierung

Mit diesem Test sollte verifiziert werden, ob sich ein Server beim Kontextrouter richtig registriert. Dazu muss sich der Server authentifizieren uns die von ihm bereitgestellten Dienste mit den jeweils unterstützten Kontexttypen auf dem Kontextrouter hinterlegen.

9.3.2.1 Durchführung

Für eine Dienstregistrierung ist zwischen Kontextrouter und Server eine sichere Verbindung via SSH-Protokoll aufzubauen. Initiiert wird dies durch den Server selbst.

Er muss über die Verbindung seine angebotenen Dienste und die dabei unterstützten Kontexttypen übermitteln. Die Ablage hat dann in einem bestimmten Verzeichnis (z. B. /home/register) zu erfolgen. Dieses Verzeichnis ist durch den Kontextrouter vorgegeben und muss bei der Konfiguration der Serverimplementierung berücksichtigt werden.

Es wurde dieselbe Messanordnung wie in Abschnitt 9.3.1 verwendet. Auch die Konfiguration der Knoten blieb gleich.

9.3.2.2 Auswertung

Bei diesem Test war zu berücksichtigen, dass Click im *Userlevel*-Modus betrieben wird, der eigentlich nur für Testzwecke entwickelt wurde. Aus diesem Grund können, wie am Ende des Abschnitts 9.3.1.2 erwähnt, Komplikationen auftreten, wenn mit dem Kontextrouter über höhere Protokolle kommuniziert wird. So wurden bei diesem Test die von der Netzwerkschnittstelle an Click übergebenen Daten zwar an das Betriebssystem weitergereicht. Die Antwort der Anwendung ist dann jedoch vom Kernel an die Netzwerkschnittstelle übergeben worden. Der Kernel versuchte, direkt (ohne Click) eine Verbindung zum anfragenden Server aufzubauen. Da sich das Ziel weiter als ein Hop vom Kontextrouter entfernt befand, konnte aber über die zum Kernel gehörende Routingtabelle keine passende Route zum Ziel gefunden werden, obwohl Click diese Information besaß. Als Folge konnte der SSH-Verbindungsaufbauwunsch nicht beantwortet werden. Alternativ ist deshalb für den hier beschriebenen Funktionstest manuell die Routeninformation übergeben worden. Damit war ebenfalls eine Dienstregistrierung per SSH über mehrere Hops möglich. Wie bereits erwähnt, sollte diese Problematik im *Kernel*-Modus nicht auftreten, weil dort die Funktionen von Click in den Kernel eingebunden sind (siehe Abschnitt 8.2.2).

Vom Server wurden während des erfolgreichen Verbindungsaufbaus folgende Ergebnisse ausgegeben:

```
ROUTER FOUND
New transmission
Router address: 10.0.0.10
Server has sent the file "503316490.list"  to "register@10.0.0.10"
Wait 540 seconds
New transmission
Router address: 10.0.0.10
Server has sent the file "503316490.list"  to "register@10.0.0.10"
Wait 540 seconds
...
```

`ROUTER FOUND` zeigt an, dass ein Advertisement erfolgreich empfangen wurde. Daraufhin wurde eine SSH-Verbindung zum Kontextrouter (IP-Adresse: 10.0.0.10) aufgebaut und die Datei 503316490.list dorthin übertragen. Der Dateiname entspricht den Konventionen aus Abschnitt 8.2.3.1 und korreliert mit dem als Server arbeitenden Knoten 2. Der Server hatte die Datei nach 540 Sekunden erneut übertragen, damit ihn der Kontextrouter nicht aus seiner Servertabelle löscht. Dieser Wert lässt sich an die Anforderungen des Kontextrouters anpassen. Er wurde hier gewählt, weil der Kontextrouter im regulären Betrieb alle Einträge aus der Tabelle löscht, die älter als 10 Minuten sind.

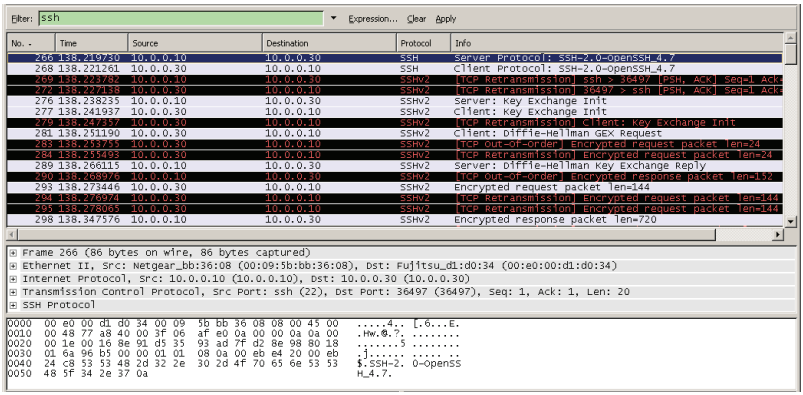


Abbildung 9.5: SSH-Verbindungsaufbau während der Registrierung

Dass die Datei tatsächlich übertragen worden ist, konnte durch eine Überprüfung des entsprechenden Verzeichnisses (aktuell: `/home/register`) nachgewiesen werden. Wie Abbildung 9.5 verdeutlicht, ist die Verbindung auch mit Hilfe von Wireshark auf dem Server nachzuweisen gewesen. In der Abbildung ist der Aufbau der ersten SSH-Verbindung zu sehen. Auffällig sind dabei die relativ häufig auftretenden Wiederholungssendungen, die beispielsweise ein Zeichen für eine schlechte Funkverbindung oder durch die parallele Funktion von Click und dem Kernel hervorgerufen sein können. Die Funktionsweise wurde dadurch nicht beeinträchtigt, sodass mit diesen Ergebnissen auch die Dienstregistrierung als erfolgreich implementiert betrachtet wird.

9.3.3 Dienstanfrage

Dieser Test diente zum Nachweis der in Abschnitt 6.3.2.4 beschriebenen Funktionen zur Anfrage eines Dienstes. Hierbei wurde auch die im Architekturkonzept vorgeschlagene optionale Funktion einer alternativen Serverauswahl verifiziert.

9.3.3.1 Durchführung

Der vorhergehende Versuchsaufbau (Abbildung 9.2) ist auch wieder bei diesem Funktionstest beibehalten worden. Knoten 2 sollte nun zur Dienstanfrage ein erweitertes AODV-RREQ-Paket an den Kontextrouter senden. Dieser musste daraufhin einen für den Client geeigneten Server anbieten. Um die Funktion der alternativen Auswahl durch den Client zu verifizieren, wurde ein zweiter Test durchgeführt. Der vorgeschlagene Server sollte hier vom Client abgelehnt und eine weitere Anfrage an den Kontextrouter gestellt werden. Der daraufhin vorgeschlagene Server war dann vom Client zu akzeptieren.

In diesem Testszenario wurde auf dem Knoten 1 AODV-UU eingesetzt, um die Interoperabilität zu einer von Click verschiedenen AODV-Implementierung zu testen. Die Anfragen durch den Client auf Knoten 2 wurden unter Verwendung der in Abschnitt 8.3.1 erwähnten Skripte mit Hilfe von Click realisiert. Gewählt wurde dabei ein fiktiver Dienst mit der Kennung 1. Das erweiterte RREQ war so konfiguriert, dass

es dem Beispiel aus Abbildung 6.10 in Abschnitt 6.3.2.5 entspricht. Die einzelnen Kontexttypen besaßen somit folgende Prioritäten:

Kontexttyp(Priorität): 1(7), 2(5), 3(4), 4(6), 5(3), 6(1), 7(2), 8(3), 9(4), 10(0)

Auf dem Kontextrouter wurden durch Ablage der entsprechenden Dateien zwei Server registriert. Da die korrekte Registrierung bereits in Abschnitt 9.3.2 nachgewiesen worden ist, konnte hier auf reale Server verzichtet werden. Die in den Registrierungsdateien enthaltenen unterstützten Kontexttypen entsprachen denen der Server S1 und S2 aus Abschnitt 6.3.2.5. Angenommen wurde dabei, dass es sich bei S1 um Knoten 1 (192.168.1.30) und bei S2 um Knoten 3 (10.0.0.20) der Demonstratoranordnung handelt. Der Kontextrouter nutzte bei der Erstellung der Serverrangliste die statische gewichtete Auswahl (siehe Abschnitt 8.2.3.3).

9.3.3.2 Auswertung

Die Anfrage des Knotens 2 zeigt Abbildung 9.6. Dort ist der Beginn des eigentlichen AODV-Paketes gekennzeichnet. Das D-Flag ist gesetzt, damit nur der Zielknoten (Kontextrouter) antwortet. Das U-Bit ist gesetzt, weil die *Destination Sequence Number* nicht bekannt ist. Schließlich ist das NR-Flag nicht gesetzt, worüber der Client dem Kontextrouter mitteilt, dass ein Rerouting erlaubt ist. Dieses Flag belegt das erste Bit im reservierten Bereich des AODV-Headers (siehe auch Abbildung 6.2 in Abschnitt 6.2.2). Der in Abbildung 9.6 enthaltene Rahmen umfasst die Daten der erweiterten RREQ-Paketstruktur. Das Feld *Type* hat entsprechend der Vereinbarung in Abschnitt 8.3.1 den Wert 16. Die Länge der Erweiterung ist aus den in Abschnitt 8.2.3.2 beschriebenen Gründen um 2 Byte größer als im Konzept beschrieben. *Session* besitzt den Wert 1, der *Identifier* entspricht dem gefundenen Server. Der *Service* sowie die Kontexttypen werden entsprechend dem Konzept aus Abschnitt 6.3.4 angezeigt.

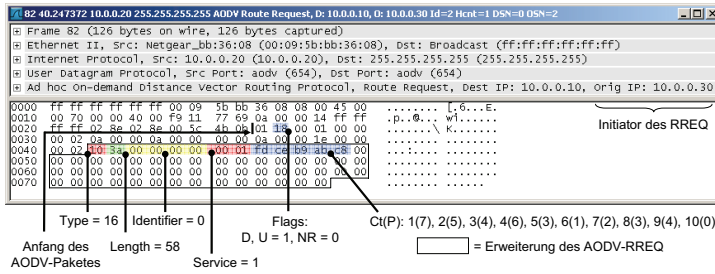


Abbildung 9.6: RREQ-Paket für eine Dienstanfrage

Im ersten Test war beim Kontextrouter nur ein für die Anfrage geeigneter Server (S1) registriert. Der Kontextrouter wertete diese mit folgenden Ergebnissen aus:

```
CServers: entry with id 192.168.1.30 inserted
CServers: found best server with x_k 36.734695 and service 1 at id
192.168.1.30 na-flag is 1
CClients: inserted service 1 at 10.0.0.30 with server 192.168.1.30
```

Die Ausgabe bestätigt die Übernahme der Registrierungsdaten in die Servertabelle des Kontextrouters. Dabei wurde mit Hilfe des Auswahlalgorithmus festgestellt,

dass die vorhandenen Kontexttypen der aktuellen Anfrage zu 36,7% mit den vom Dienst unterstützten Kontexttypen übereinstimmen. Dies entspricht genau dem berechneten Wert aus Tabelle 6.5. Weiterhin ist der angeforderte Dienst (`service 1`) und der zugehörige Server ausgegeben worden. Schließlich wurde das NA-Flag für das Antwortpaket gesetzt, da es keine weiteren Server gegeben hatte, die diesen Dienst anbieten. Die Daten unter `CClients` wurden in die zu den Clients gehörenden Tabellen des Kontextrouters eingetragen. Die Ausgabe von Wireshark hat die vom Kontextrouter ausgegebenen Informationen bestätigt. In Abbildung 9.7 ist dazu das Antwortpaket des Kontextrouters dargestellt. Das Feld *Type* der RREP-Erweiterung ist hier mit Rücksicht auf die Vereinbarung in Abschnitt 8.3.1 mit 17 belegt. Die Länge der Paketerweiterung beträgt 22. Das ist aus den in Abschnitt 8.2.3.2 beschriebenen Gründen ein Byte mehr als im Konzept vorgeschlagen. Das NA-Flag ist, wie erwartet, gesetzt. Das Flag belegt das erste im AODV-RREP-Paket reservierte Bit (siehe Abbildung 6.2 in Abschnitt 6.2.2). Die Sitzungsnummer (*Session*) hat den Wert 1 und der *Identifier* entspricht dem ausgewählten Server. Darüber hinaus sind alle vom Server S1 unterstützten Kontexttypen abgebildet.

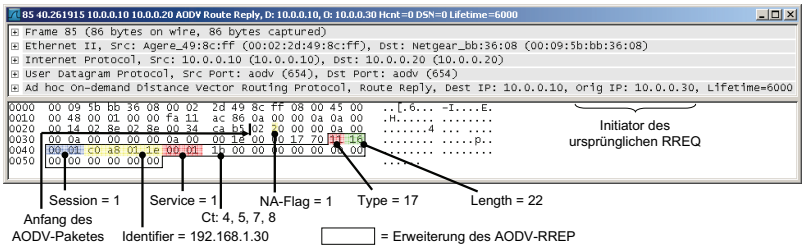


Abbildung 9.7: RREP-Paket des Kontextrouters

Im zweiten Test wurden nun die Registrierungsdateien beider Server auf dem Kontextrouter hinterlegt. Um die Tabellen des Kontextrouters zurückzusetzen, ist dieser außerdem neu gestartet worden. Dies war nötig, da die Serverauswahl mit der statischen Rangliste erfolgte und diese erst nach einem spezifischen Timeout aktualisiert wird. Die Ausgabe des Kontextrouters bestätigte nun wieder die Übernahme der Daten in die Servertabelle. Der beste Server (S2) wurde ausgewählt. Er besaß eine 51%ige Übereinstimmung in den Kontexttypen. Das NA-Flag war nun nicht gesetzt. Somit wurde dem Client angezeigt, dass bei Bedarf eine Alternative zur Verfügung steht. Im Einzelnen gestaltete sich die Ausgabe wie folgt:

```
CServers: entry with id 192.168.1.30 inserted
CServers: entry with id 10.0.0.20 inserted
CServers: found best server with x_k 51.020409 and service 1 at id
10.0.0.20 na-flag is 0
CClients: inserted service 1 at 10.0.0.30 with server 10.0.0.20
```

Die Ergebnisse werden auch durch Abbildung 9.8 bestätigt. Dort ist das aufgezeichnete Antwortpaket des Kontextrouters dargestellt. Das NA-Flag ist, wie bereits erwähnt, dieses Mal nicht gesetzt. Die weitere Belegung der Paketerweiterung entspricht im Wesentlichen der aus Abbildung 9.7. Allerdings werden nun der zum S2 gehörende *Identifier* und die von diesem Server unterstützten Kontexttypen angezeigt.

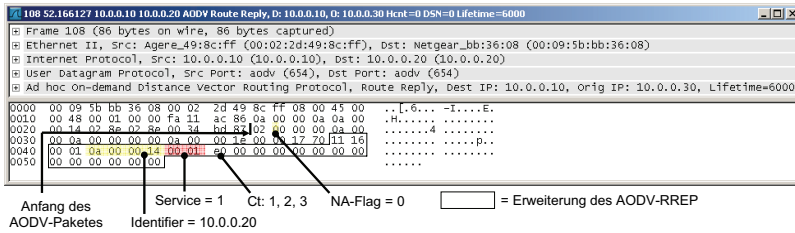


Abbildung 9.8: RREP-Paket nach erster Anfrage durch den Client

Der Client musste den gefundenen Server nach Empfang des RREP ablehnen und dafür erneut eine Anfrage senden. Laut Konzept war dabei der *Identifier* des letzten RREP-Paketes mit zu übertragen. Ein auf dieses RREP folgendes RREQ-Paket ist in Abbildung 9.9 dargestellt. Im Gegensatz zu Abbildung 9.6 hat der *Identifier* nun den Wert des zuletzt vorgeschlagenen Servers. Die anderen Werte der AODV-Paketerweiterung sind unverändert.

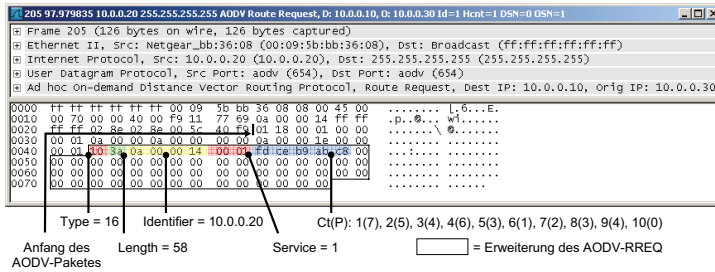


Abbildung 9.9: Wiederholtes erweitertes RREQ-Paket vom Client

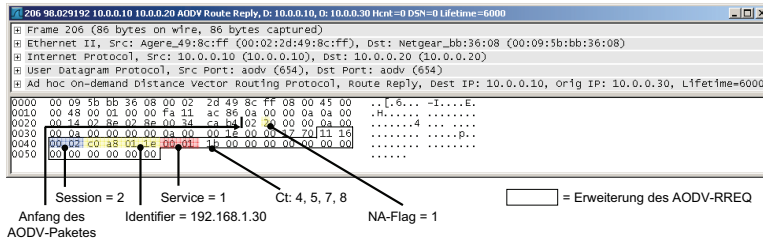


Abbildung 9.10: RREP-Paket nach wiederholter Anfrage durch den Client

Nach dem Empfang des zweiten RREQs wählte der Kontextrouter den nächsten Server (S1) in der Rangliste aus und sendete, wie in Abbildung 9.10 dargestellt, ein RREP zurück. Dort wurde für die anschließende Kommunikation ein neuer Wert für *Session* und der aktualisierte *Identifier* übertragen. Des Weiteren waren nun die vom Server S1 unterstützten Kontexttypen enthalten. Das NA-Flag war gesetzt, wodurch dem Client angezeigt wurde, dass es keine weitere Server gibt, die diesen Dienst unterstützen.

Mit den vorangegangenen Tests konnte die korrekte Struktur der erweiterten RREQ- und RREP-Pakete und die richtige Funktionsweise des Kontextrouters anhand einer Auswahlfunktion erfolgreich nachgewiesen werden. Die Kommunikation erfolgte dabei über zwei Hops. Das Verfahren ist damit für Ad-hoc-Netze geeignet. Dabei arbeitete es mit der Implementierung AODV-UU zusammen. Das führt zu der Schlussfolgerung, dass auch die regulären AODV-Funktionen des Kontextrouters nach Standard arbeiten.

9.3.4 Weiterleitungsfunktion

Nach einer Dienstregistrierung kann ein Client mit dem gefundenen Server kommunizieren. Der Kontextrouter leitet dazu die zugehörigen Daten entsprechend Abschnitt 6.3.2.6 weiter. Dazu muss der *Identifier* eines ankommenden IP-Paketes ausgetauscht und das modifizierte Paket an den gewünschten Zielserver weitergesendet werden. Herkömmlicher IP-Verkehr ist vom Kontextrouter dagegen konventionell zu routen.

9.3.4.1 Durchführung

Um die Weiterleitungsfunktion zu verifizieren, wurde die Testanordnung aus Abbildung 9.11 verwendet. Bei diesem Test konnte darauf verzichtet werden, Registrierungsdateien von Servern beim Kontextrouter abzulegen, weil der Router alle IP-Pakete weiterleitet, sofern diese die entsprechenden Erweiterungen besitzen und der *Identifier* eine IP-Adresse repräsentiert, deren Route bekannt ist. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass das Konzept des kontextsensitiven Routings nicht vorschreibt, dass als *Identifier* die IP-Adresse des in Frage kommenden Zielservers verwendet werden muss. Es können auch andere eindeutige Kennungen genutzt werden. Allerdings erschwert dies die Nutzung einer Blacklist durch den Client.

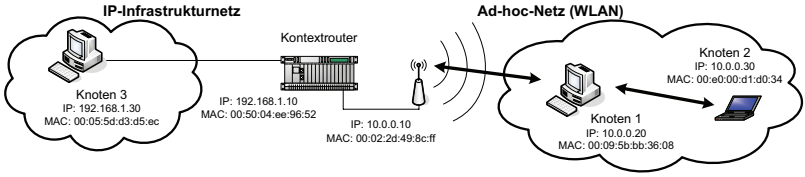


Abbildung 9.11: Anordnung für Weiterleitungstest

Zum Test wurde vom Knoten 2 aus ein speziell präpariertes Ping-Paket (ICMP-*Echo-Request*) entsprechend dem in Abschnitt 6.3.2.6 vorgestellten Konzept zur Weiterleitung von IP-Paketen gesendet. Dazu sind die in Abschnitt 8.3.1 erwähnten Click-Skripte verwendet worden. Im Testszenario befand sich der Knoten 2 außer Reichweite des Kontextrouters, sodass Knoten 1 zur Weiterleitung der Pakete genutzt wurde. Dieser Knoten arbeitete dabei wieder mit AODV-UU. Zur Weiterleitung der Pakete musste der Kontextrouter deren IP-Header entsprechend dem Konzept modifizieren und an den Zielserver (Knoten 3) senden. Dieser Knoten sollte dann ebenfalls mit einem für das kontextsensitive Routing präparierten ICMP-*Echo-Reply*-Paket

antworten. Die Antwort war wieder an den Kontextrouter zu übertragen, wo der IP-Header entsprechend dem Konzept modifiziert und das Paket über Knoten 1 an Knoten 2 gesendet werden sollte.

Neben der Weiterleitungsfunktion wurde mit dieser Anordnung auch getestet, ob die herkömmliche IP-Kommunikation über Netzgrenzen hinweg funktioniert. Dazu ist mit Hilfe eines herkömmlichen Ping versucht worden, Knoten 3 von Knoten 2 aus zu erreichen und umgekehrt. Einzige Voraussetzung dafür war, dass bei den miteinander kommunizierenden Knoten 2 und 3 der Kontextrouter als *Default-Gateway* eingetragen ist, da es den Betriebssystemen sonst nicht möglich ist, Zieladresse und Netzwerkschnittstelle richtig zuzuordnen.

9.3.4.2 Auswertung

Die richtige Funktionsweise konnte wieder mit Hilfe von Wireshark nachgewiesen werden. Abbildung 9.12 zeigt dazu die Ausgabe der ICMP-Pakete an den Schnittstellen, die den Zugang zu den an der Kommunikation beteiligten Netzwerken bilden. Im oberen Teil der Abbildung befindet sich die Ausgabe für das Ad-hoc-Netz und im unteren Teil die des Ethernets. Anhand der Zeitstempel ist ersichtlich, dass Knoten 2 (IP: 10.0.0.30) ein ICMP-*Echo-Request* über Knoten 1 an den Kontextrouter (IP: 10.0.0.10) gesendet hat. Dieser leitete es über das Ethernet-Interface (IP: 192.168.1.10) an den Knoten 3 (IP: 192.168.1.30) weiter. Die Antwort in umgekehrter Richtung kann mit dem ICMP-*Echo-Reply* nachverfolgt werden. Dieses erscheint in Abbildung 9.12 jeweils zweimal an der Ad-hoc-Netz-Schnittstelle, da Wireshark auch die Übermittlung von Knoten 1 zu Knoten 2 registriert hatte.

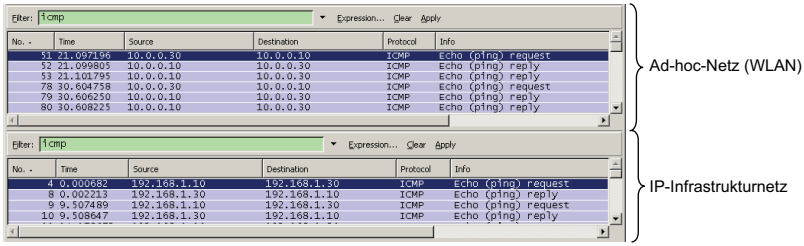


Abbildung 9.12: Weiterleitung von Paketen beim Kontextrouter

Eine weitere Analyse der markierten Pakete zeigt Abbildung 9.13. Dort stimmt der Aufbau des *Options*-Feldes von IPv4 mit dem Vorschlag aus Abschnitt 6.3.2.6 überein. Das *Copied Flag* ist gesetzt, die *Option Class* hat den Wert 0 und die *Option Number* den Wert 30. Die Länge des *Options*-Feldes beträgt 8 Bytes. In diesem Fall hat die *Session* die Kennung 1. Diese ist, wie vorgeschrieben, bei beiden Paketen gleich. Bei den zu Knoten 2 gehörenden ICMP-*Echo-Requests* ist der auf Knoten 3 mappende *Identifier* enthalten. Dieser wurde vom Kontextrouter ausgetauscht und durch den auf den Knoten 2 mappenden *Identifier* ersetzt. Anhand der Protokolldaten kann auch nachvollzogen werden, dass die ICMP-Daten in beiden Fällen identisch waren.

Abschließend wurde die Übertragung mit herkömmlichen IP-Verkehr getestet. Dazu wurde das Click-Skript auf dem Knoten 2 so modifiziert, dass „normale“ ICMP-*Echo-*

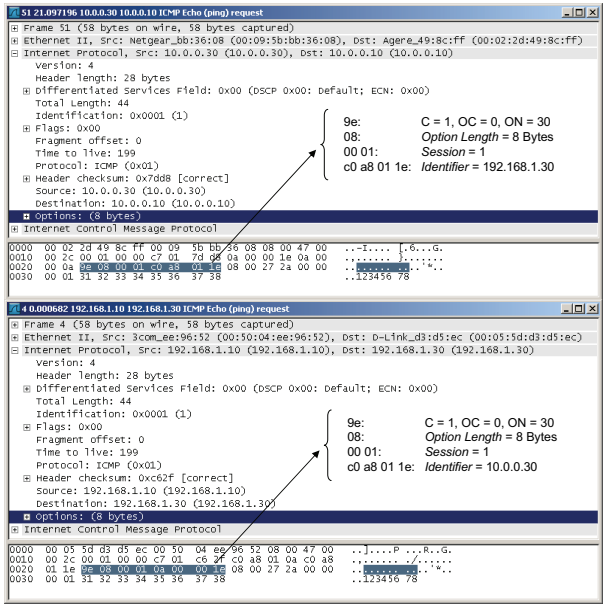


Abbildung 9.13: Das *Option*-Feld im modifizierten ICMP-*Echo-Request*

Request-Pakete versendet werden konnten. Die Übertragung erfolgte problemlos. In Abbildung 9.14 sind IP-Quelle und -Ziel zu sehen. Außerdem ist wieder anhand der MAC-Adressen zu erkennen, dass die Kommunikation über Knoten 1 und den Kontextrouter erfolgte.

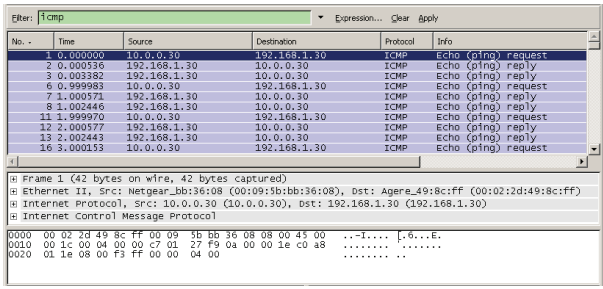


Abbildung 9.14: Versand herkömmlicher Ping-Pakete

Der Kontextrouter erfüllt damit alle an ihn gestellten Vermittlungsaufgaben. Da der Test mit Ping-Paketen eine Kommunikation in beide Richtungen erfordert, ist damit bewiesen, dass die Kommunikation unabhängig vom Netzwerktyp, d. h. in heterogenen Netzwerkumgebungen, funktioniert. Bei einer Anfrage vom Ethernet aus muss er dazu allerdings die Routensuche im Ad-hoc-Netz übernehmen.

9.3.5 Rerouting

Fällt ein Server während der aktuellen Dienstkommunikation aus, ist der Kontextrouter in der Lage den zugehörigen Verkehr an einen alternativen Server weiterzuleiten. Der folgende Test diente zum Nachweis dieser Funktion.

9.3.5.1 Durchführung

Das Verifizieren des Reroutings erfolgte mit dem Versuchsaufbau aus Abbildung 9.15. Die in diesem Szenario eingesetzte Software entspricht derjenigen aus Abschnitt 9.3.4. Der Kontextrouter war so konfiguriert, dass zur Bestimmung eines geeigneten Servers der Algorithmus für die gewichtete Auswahl genutzt wird. Die vom alternativen Server unterstützten Kontexttypen konnten dabei beliebig von denen des ursprünglichen Servers abweichen. In der dynamischen Variante des Auswahlalgorithmus kann diese Abweichung auch begrenzt werden.

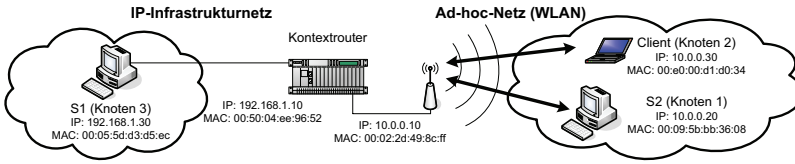


Abbildung 9.15: Anordnung für Rerouting-Test

Knoten 2 arbeitete bei diesem Test als Client und war bereits im Besitz der Router-IP-Adresse. Er sollte eine Dienstanfrage starten und dabei dem Kontextrouter mitteilen, dass ein Rerouting erlaubt ist (NR-Flag ist nicht gesetzt). Der Dienst und die vom Client bereitgestellten Kontexttypen sowie deren Priorisierung entsprachen den Vorgaben aus der Dienstanfrage in Abschnitt 9.3.3. Das Gleiche gilt für die Registrierungsdateien der Server S1 und S2, die bereits auf dem Kontextrouter abgelegt waren. Die weitere Kommunikation sollte dann entsprechend dem Konzept erfolgen. Danach muss der Kontextrouter nach Empfang des erweiterten RREQs S2 als den geeignetsten Server für den Client identifizieren. Nach einer entsprechenden Antwort des Kontextrouters können dann Client und Server miteinander kommunizieren. Dazu sendet der Client entsprechend modifizierte Ping-Pakete. Während dieser Kommunikation verlässt dann S2 das Ad-hoc-Netz bzw. gerät außer Reichweite. Entsprechend dem Konzept von AODV (RFC 3561), versucht der Kontextrouter daraufhin, eine neue Route zu dem Server zu finden. Die Anzahl der Versuche ist dabei begrenzt. Wird keine Route gefunden, leitet der Router alle Anfragen an einen alternativen Server – den nächsten in der Serverrangliste – weiter.

9.3.5.2 Auswertung

Während der Versuchsdurchführung konnte über die folgenden Ausgaben des Kontextrouters der Reroutingprozess analysiert werden:

```
CServers: found best server with x_k 51.020409 and service 1 at id 10.0.0.20 na-
flag is 0
CClients: inserted service 1 at 10.0.0.30 with server 10.0.0.20
CClients: found entry with id 10.0.0.30 and sid 1
```

```
CIPForward: entry found, forwarding to server 10.0.0.20
CClients: found entry with id 10.0.0.30 and sid 1
CIPForward: entry found, forwarding to client 10.0.0.30
CClients: found entry with id 10.0.0.30 and sid 1
...
CIPForward: entry found, forwarding to server 10.0.0.20
CServers: entry with 10.0.0.20 deleted
cservers: file /home/register/335544330.list deleted
CClients: found entry with id 10.0.0.30 and sid 1
...
CIPForward: found alternative server 192.168.1.30, forwarding packet with src
10.0.0.10 dst 192.168.1.30 id 10.0.0.30 to network
CClients: update - delete ls 10.0.0.20, insert ns 192.168.1.30 at entry 10.0.0.30
CClients: found entry with id 10.0.0.30 and sid 1
CIPForward: entry found, forwarding to client 10.0.0.30
CClients: found entry with id 10.0.0.30 and sid 1
```

Bei der Anfrage des Clients wurde Server S2 (`id 10.0.0.20`) ausgewählt und in die zu den Servern und Clients zugehörigen Tabellen eingetragen. Danach erhielt der Kontextrouter ein für das kontextsensitive Routing modifiziertes Ping-Paket vom Client, überprüfte die entsprechende Tabelle in `CClients` und leitete dann das Paket an S2 weiter. Die Antwort wurde dementsprechend von S2 an den Client weitergeleitet. Dann „verließ“ S2 das Ad-hoc-Netz. Der Kontextrouter konnte deshalb keine ICMP-*Echo-Request*-Pakete mehr an ihn übergeben. Nachdem auch nach mehreren AODV-RREQ-Anfragen keine Route zu S2 gefunden worden war, leitete der Kontextrouter das Rerouting ein. Die zum S2 gehörenden Einträge in der Server-Routing-Tabelle und die entsprechende Registrierungsdatei wurden gelöscht. Der Kontextrouter griff daraufhin auf den zweiten Server in der Rangliste zu (`alternative server 192.168.1.30`) und aktualisierte alle zugehörigen Tabellen. Dadurch konnten die Daten jetzt an den Server S1 weitergeleitet werden. Wie im Konzept gefordert, blieb hierbei die Kennung der *Session* (`sid 1`) gleich. Das Rerouting war also erfolgreich.

In Abbildung 9.16 ist das erste erfolgreiche Antwortpaket nach dem Rerouting für beide Schnittstellen dargestellt. Die Sequencenummern stimmen in beiden Fällen überein (`Sequence number: 10`). Die Erweiterung des IP-Paketes für die Kommunikation zwischen S1 und Kontextrouter beinhaltet den *Identifier* für den Client (`10.0.0.30`). Das Paket an der Schnittstelle zum Ad-hoc-Netz übermittelt dagegen den ursprünglichen zum Server S2 gehörenden *Identifier* (`10.0.0.20`). Der Client hatte entsprechend dem Konzept nichts von einem Wechsel des Servers bemerkt. Da das Rerouting zwischen verschiedenen Netzwerktypen stattfand, ist auch hier wieder die Interoperabilität des Verfahrens nachgewiesen.

Der aktuelle Ansatz des Rerouting ist für Ad-hoc-Netze sehr gut geeignet und wurde für die dortigen Anforderungen auch eingeführt. Im Infrastrukturnetz müssen für ein erfolgreiches Rerouting zusätzliche Maßnahmen getroffen werden. Aktuell kann sich ein Server dort beim Kontextrouter über die Registrierungsdatei abmelden. Wird also beim Rerouting ein Auswahlalgorithmus mit dynamischer Rangliste genutzt, kann während des Reroutingprozesses festgestellt werden, ob der alternative Server noch anwesend ist. Allerdings wird ein Ausfall während einer laufenden Kommunikation

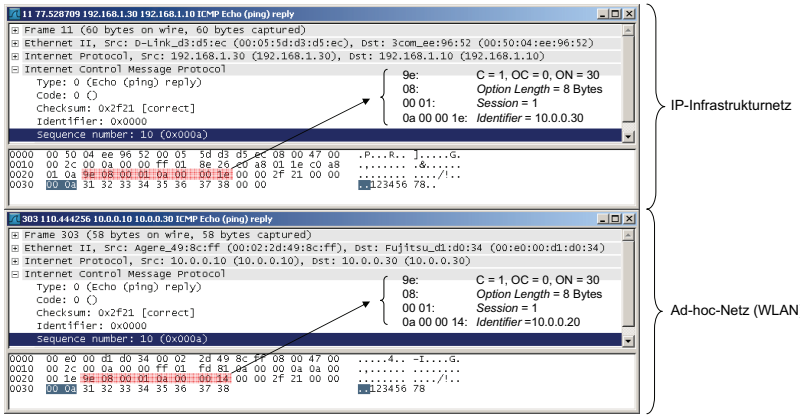


Abbildung 9.16: Rerouting von Server S2 zu Server S1

mit diesem Server nicht bemerkt. Dies könnte in einer weiteren Entwicklungsstufe verbessert werden, indem der Server vor Verlassen des Netzwerkes dem Kontextrouter beispielsweise mit Hilfe von entsprechenden Nachrichten mitteilt, dass nun sämtliche Routertabellen, die diesen Vorgang betreffen, aktualisiert werden müssen.

9.3.6 AODV-Funktionalität

Der Kontextrouter soll sich nach dem Architekturkonzept in einem Ad-hoc-Netz wie ein mobiler Knoten verhalten. Deshalb war zum Abschluss der Funktionstests noch die Funktion des AODV bei einem horizontalen Handover (siehe Abschnitt 3.2) zu verifizieren.

9.3.6.1 Durchführung

Für den Nachweis der AODV-Funktionalität erfolgten zwei Tests. Hierbei wurde wieder die Versuchsanordnung aus Abbildung 9.2 verwendet. Im ersten Testszenario kommunizierte Knoten 1 mit dem Kontextrouter. Knoten 2 befand sich diesmal allerdings zuerst direkt in Reichweite von Knoten 1 und Kontextrouter. Er wurde dann vom Router weg bewegt, blieb aber während des gesamten Tests in Reichweite von Knoten 1. War der Abstand zwischen Knoten 2 und Kontextrouter so groß geworden, dass keine direkte Kommunikation mehr möglich war, musste folglich ein Handover auf Knoten 1 stattfinden. Für das zweite Testszenario haben die gleichen Ausgangsbedingungen gegolten, allerdings kommunizierten diesmal Knoten 1 und 2 direkt miteinander. Sobald sich dann beide außer Reichweite bewegten, sollte der Kontextrouter als Zwischenknoten dienen.

Zur Realisierung der beiden Szenarien wurde auf beiden Knoten AODV-UU eingesetzt. Die Konnektivität zwischen den jeweiligen Knoten ist mit Ping getestet worden. Wireshark arbeitete auf dem Knoten 2, was an dieser Stelle von Bedeutung ist, da es die angezeigten duplizierten ICMP-Pakete erklärt.

9.3.6.2 Auswertung

Wie die Tests zeigten, gliedert sich der Kontextrouter vollständig in ein Ad-hoc-Netz ein. Abbildung 9.17 illustriert dazu die Paketdaten für das erste Testszenario. Im oberen Ausgabefenster von Wireshark ist das letzte ICMP-*Echo-Reply*-Paket der direkten Kommunikation zwischen Knoten 2 und Kontextrouter zu sehen. Nach diesem Zeitpunkt bewegte sich Knoten 2 außer Reichweite des Kontextrouters und die Route ging verloren. Der Knoten 2 versuchte nun den Kontextrouter über Konten 1 zu finden. Zu sehen ist dies an der Ziel-MAC-Adresse des ICMP-*Echo-Request*-Pakets (Nr. 664, mittlere Paketdarstellung in Abbildung 9.17), die zum Knoten 1 gehört. Die Ziel-IP-Adresse ist hierbei gleich geblieben. Das erste darauf folgende erfolgreiche ICMP-*Echo-Reply*-Paket ist das Paket Nr. 681, dessen Inhalt in Abbildung 9.17 unten dargestellt ist. Der Abstand zwischen Anfrage und Antwort beträgt dabei ca. 3,6 s. In dieser Zeit wurden 4 unbeantwortete ICMP-*Echo-Requests* gesendet. Bei den anderen angezeigten *Requests* handelte es sich um die „mitgehörten“ Anfragen vom Zwischenknoten zum Kontextrouter.

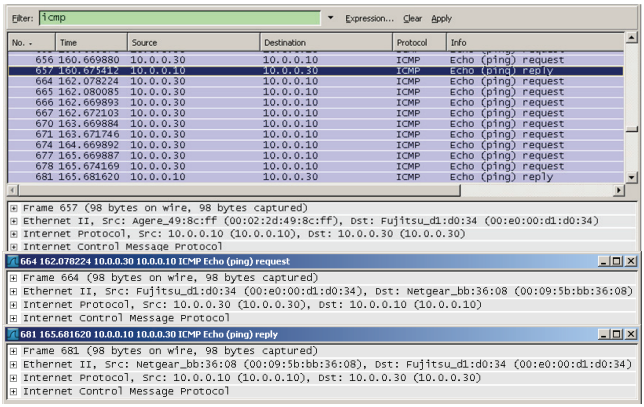


Abbildung 9.17: Handover beim ersten Szenario

Im zweiten Testszenario – Kontextrouter als Zwischenknoten – ergab sich das gleiche Verhalten. In Abbildung 9.18 ist dazu ein Spezialfall dargestellt, bei dem die Kommunikation während des Handovers über verschiedene Wege erfolgte. Dort ist anhand der MAC-Adressen zu sehen, dass das ICMP-*Echo-Request* zwar über den Kontextrouter weitergeleitet worden ist, Knoten 1 aber direkt dem Knoten 2 geantwortet hat. Dieser Effekt trat bei dieser Messung deshalb verstärkt auf, weil die Sendeleistung der Netzwerkkarte am Knoten 1 größer war als beim Kontextrouter und beim Knoten 2. D. h., dass in der Phase eines Handovers alle Geräte zwar Daten vom Knoten 2 direkt empfangen, aber nicht direkt an ihn senden konnten. Bei Knoten 1 wurden also alle beteiligten Geräte als direkte Nachbarn erkannt, während bei Knoten 2 nur der Kontextrouter als unmittelbarer Nachbar zur Verfügung stand. Dass es sich um die zur Anfrage gehörende Antwort handelte, ist in Abbildung 9.18 an der identischen Sequenznummer zu erkennen.

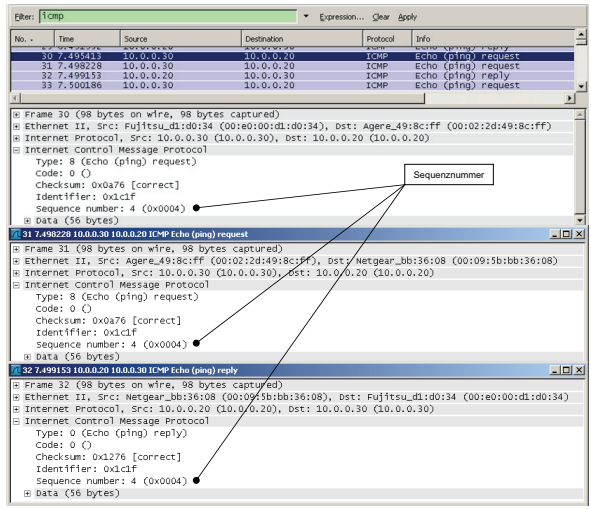


Abbildung 9.18: Handover beim zweiten Szenario

9.4 Performancetest

Die im Rahmen des Perfomancetest durchgeführten Messungen sollten erste Aussagen über die Leistungsfähigkeit des Kontextrouters während des Reroutings liefern.

9.4.1 Durchführung

Für die Messungen wurde die Versuchsanordnung aus Abbildung 9.15 genutzt. Dabei wurde mit Hilfe von Wireshark die Zeit gemessen, die zwischen dem Ausfall eines Servers bis zum Finden eines alternativen Servers verging. Die Messung erfolgte an der WLAN-Schnittstelle des Kontextrouters. Zur Aufnahme der Zeit diente der Abstand zwischen erstem ankommenden ICMP-*Echo-Request*-Paket, welches den Server S2 nicht mehr erreichen konnte und dem ersten erfolgreich vom alternativen Server empfangenen ICMP-*Echo-Reply*-Paket. Um die Zeit zu erhalten, die allein für den Reroutingprozess nötig ist, musste die Dauer, die das Ping-Paket vom Kontextrouter zum alternativen Server und wieder zurück benötigt hat, abgezogen werden.

9.4.2 Auswertung

Insgesamt wurden 60 Messungen durchgeführt (siehe Anhang F). Die einzelnen Werte schwankten zwischen 7,66 s und 8,51 s. Der Durchschnitt betrug 8,16 s. Diese Werte erscheinen im ersten Moment als sehr hoch. Zu berücksichtigen ist dabei jedoch, dass sich der Kontextrouter innerhalb eines Ad-hoc-Netzes bei Routenverlust, wie in Abschnitt 9.3.5 erläutert, nach RFC 3561 verhält. So wird per Default dreimal versucht, eine neue Route zu einem Netzknoten zu finden. Bei jedem Versuch muss eine bestimmte Zeit auf eine eventuelle Antwort gewartet werden, bevor ein weiteres RREQ-Paket gesendet werden darf oder aber ein Rerouting eingeleitet wird. Unter diesen Voraussetzungen ist zukünftig noch eine Optimierung des Reroutingprozesses möglich. Allerdings können dabei die Anzahl und die zeitlichen Abstände

der Versuche, eine neue Route zu finden, nicht beliebig minimiert werden. Aufgrund der stetigen Topologieänderung würde sich dann gerade in mobilen Netzwerken die Anzahl der Reroutingprozesse stark erhöhen. Hinzu kommt, dass der Optimierungsprozess auch von der jeweiligen Netzgröße abhängig ist.

Zu beachten ist außerdem, dass sich der alternative Server bei der aktuellen Messung im IP-Infrastrukturnetz befand. Sobald sich dieser ebenfalls im Ad-hoc-Netz aufhält, können sich die gemessenen Zeiten möglicherweise weiter erhöhen. Es muss dann zusätzlich erst noch eine Wegesuche zum alternativen Server eingeleitet werden. Spürbar wird dies aber erst, wenn der Server über mehrere Hops erreichbar und die Netztopologie sehr dynamisch ist.

Eine weitere Möglichkeit, diesbezüglich die Reaktionszeit beim Rerouting zu verringern, besteht darin, mehrere Routen zu gleichwertigen Servern gleichzeitig zu betreiben oder zumindest für die Dauer der Kommunikation „offen“ zu halten. Sobald dann der kommunizierende Server nicht mehr erreichbar ist, können die vorgehaltenen Routen sofort genutzt werden. Sicherlich gibt es diesbezüglich noch eine Reihe weiterer Ansätze (z. B. parallele Routen zu demselben Server), die aber Thema zukünftiger Forschungen sein müssen. Einen ersten Eindruck über die dabei zu berücksichtigenden Probleme hat der durchgeführte Performancetest aufgezeigt. Weitere Betrachtungen gehen jedoch über den Rahmen dieser Arbeit hinaus.

9.5 Kapitelzusammenfassung

Die vorliegenden Funktionstests sind konzeptioniert worden, um das in Kapitel 6 vorgeschlagene Architekturkonzept und insbesondere den Kontextrouter als Kernelement testen zu können. Die während der Durchführung verwendeten Versuchsanordnungen und Hilfsmittel wurden beschrieben, sodass alle Ergebnisse reproduzierbar sind. Die Funktionstests wurden auf Versuche im WLAN-Ad-hoc-Netz und dem IP-Infrastrukturnetz beschränkt, weil das beim Demonstrator auf Basis des Ethernet realisierte Ad-hoc-Netz diesbezüglich bereits in [Wenz07a] geprüft wurde.

Somit konnte nun für den gesamten Demonstrator nachgewiesen werden, dass der Kontextrouter die an ihn gestellten und im Konzept aufgeführten Anforderungen erfüllt. Dazu gehören:

- das Versenden von Advertisements und die Reaktion auf Solicitations.
- das Registrieren von Diensten.
- die Bearbeitung von Dienstanfragen.
- die Unterstützung einer alternativen Serverwahl durch den Client.
- das Routing bzw. die Weiterleitung von für kontextsensitive Dienste modifiziertem IP-Verkehr.
- das Routing bzw. die Weiterleitung von herkömmlichem IP-Verkehr.
- das Rerouting bei einem Serverausfall.
- der Einsatz als mobiler Knoten im Ad-hoc-Netz.

Die Tests fanden in der Regel über mehrere Hops im AODV-Netz statt. Dabei wurde auch die Interoperabilität zu AODV-UU erfolgreich getestet. Schließlich konnte mit den vorliegenden Tests auch nachgewiesen werden, dass das System in heterogenen Netzwerkumgebungen funktioniert. Der Aufbau und die Erweiterung der im Konzept vorgeschlagenen Pakete haben sich bei allen Tests bewährt und sind zu den herkömmlichen Protokollen kompatibel.

Mit dem Performance-Test konnte gezeigt werden, wie sich das System unter den Vorgaben von AODV (RFC 3561) verhält, nachdem ein Netzknoten nicht mehr erreichbar ist. Hier besteht noch Bedarf an weiteren Forschungen, um die Zeiten für den Reroutingprozess zu verbessern. Dazu wurden erste Lösungsansätze vorgestellt. So kann beispielsweise das Verhalten von AODV für den Fall einer erfolglosen Routensuche modifiziert werden. Eine weitere Variante wäre die gleichzeitige Nutzung bzw. Aufrechterhaltung von Routen zu mehreren Servern.

Während der Tests sind auch kleinere Probleme aufgetreten, die letztlich aber nicht das Konzept des kontextsensitiven Routings beeinträchtigen, sondern auf den Betrieb des Routers im *Userlevel*-Modus zurückzuführen sind. Folgende wesentliche Erfahrungen wurden gesammelt:

- Pings vom Kernel selbst werden an Click nicht übergeben. Deshalb besitzen die Click-Skripte eine eigene Funktion zur Bearbeitung von Ping-Paketen. Außerdem ist bei Tests mit Ping diese Funktion im Linux-Kernel zu unterbinden, da die Ergebnisse sonst durch Duplikate verfälscht werden.
- Es ist zu berücksichtigen, dass der Kontextrouter im *Userlevel*-Modus arbeitet. Kommunikationsdaten von Anwendungsprotokollen (z. B. SSH) übergibt der Kernel jedoch direkt an die Netzwerkschnittstelle. Die vom Kernel verwaltete Routingtabelle ist mit der des AODV jedoch nicht identisch, sodass eine Kommunikation über mehrere Hops fehlschlägt.
- Die Eigenschaften des Ad-hoc-Netzes waren stark von den Umgebungsparametern abhängig. Gerade für zukünftige Verkehrsmessungen wird es deshalb notwendig sein, eine Umgebung zu schaffen, die von äußeren Einflüssen (z. B. durch weitere WLANs) weitestgehend frei ist.
- Die Performancetests sind stark von der verwendeten Hardware abhängig. Zukünftig können bei solchen Messungen schon deshalb bessere Ergebnisse erwartet werden, weil sich die Bitrate durch den Einsatz von WLAN-Netzwerkkarten, die die neuesten Spezifikationen umsetzen, erhöhen lässt. Der Kontextrouter unterstützt gegenwärtig nur 11 MBit/s. Die Daten der anderen Netzknoten sind in Anhang E enthalten.

Die Tests haben alle Hauptfunktionen der kontextsensitiven Routingarchitektur erfolgreich nachgewiesen. Natürlich sind darüber hinaus noch viele weitere Versuche möglich und für die weitere Entwicklung unumgänglich. Allerdings zeigen die Ergebnisse, dass der vorgestellte Demonstrator eine solide Basis dafür bilden kann. Er stellt darüber hinaus auch ein flexibles Mittel zum Test von realen kontextsensitiven Diensten dar.

10. Ausblick

Die vorgestellte Architektur für das kontextsensitive Routing bildet eine solide Grundlage für zukünftige Arbeiten und Forschungen. Um das Konzept in einer IPv6-Umgebung einsetzen zu können, muss lediglich eine Anpassung der betroffenen Protokolle erfolgen.

Zur Ergänzung sollte die Demonstratorumgebung in der nächsten Entwicklungsphase multicastfähig gestaltet werden. Für die Routingarchitektur ist deshalb eine Multicastkommunikation zu bevorzugen, weil Ad-hoc-Netze in der Regel keine Weiterleitung von Teilnehmerdaten via herkömmlichen Broadcast unterstützen. Die Multicastunterstützung bildet auch die Voraussetzung für eine Umstellung des Demonstrators auf IPv6.

Vor einem produktiven Einsatz sind Simulationen und zusätzliche Tests notwendig. Dazu ist der Demonstrator weiter auszubauen. Mit Hilfe von Virtualisierungssoftware können dabei mehrere Endgeräte auf einer Hardwareplattform realisiert werden, sodass zukünftig auch sehr große Testnetzwerke realisierbar sind. Für Simulationen bietet sich die ns-2-Schnittstelle zur Routersoftware Click an.

Einen weiteren Schwerpunkt der zukünftigen Forschungsarbeit sollte der Bereich des Rerouting bilden. So ist beispielsweise die Dauer des Reroutingprozesses noch zu optimieren. Diesbezügliche Lösungsansätze wurden in Abschnitt 9.4 vorgestellt. Aber auch im Zusammenspiel mit der Möglichkeit der Ablehnung eines Servers durch den Client, ist die Kommunikationsprozedur weiter zu entwickeln. Eine mögliche Variante könnte darin bestehen, dass der Client zu Beginn im RREQ eine Liste mit unerwünschten Servern an den Kontextrouter mit übermittelt und dort auch angibt, inwieweit im Falle eines Rerouting die vom Dienst unterstützen Kontexttypen eines alternativen Servers vom ursprünglichen Angebot abweichen dürfen. Genauso könnte dem Client auch die Möglichkeit gegeben werden, zu entscheiden, welcher Algorithmus zur Auswahl eines Servers verwendet werden soll.

Es ist auch weiter zu untersuchen, inwieweit die Konfigurationsparamter des Kontextrouters (z. B. Zeiten für Alterung der Tabellen) optimiert werden können. Dazu ließe sich das Konzept um vielfältige Funktionen erweitern. Beispielsweise könnten

den Clients und Servern innerhalb eines Advertisement-Paketes alternative Kontextrouter vorgeschlagen werden. Darüber hinaus könnte auch eine Kommunikation zwischen verschiedenen Kontextroutern helfen, Informationen über verfügbare Server auszutauschen. Damit werden auch Algorithmen zur Last- und Funktionsverteilung realisierbar. Bei Überlastung eines Kontextrouters wäre es dann möglich, Anfragen von Clients an andere alternative Kontextrouter weiterzuleiten. Aber auch innerhalb des Kontextrouters sollten Mechanismen bzw. Algorithmen vorgesehen werden, die eine ausgewogene Verteilung der Last unter den registrierten Servern unterstützen. Ähnlich dem DNS könnte zusätzlich über eine Kaskadierung oder Baumtopologie eine Anfrage, die von einem Kontextrouter nicht beantwortet werden kann, an einen Nachbarrouter weitergegeben werden. Aktuell werden die Informationen über einen Kontextrouter nur innerhalb eines an diesem angeschlossenen IP-Subnetzes bzw. des Ad-hoc-Netzes verbreitet. Soll eine Verbreitung über Netzwerkgrenzen hinweg erfolgen, könnten als Relay-Agenten arbeitende Netzknoten bzw. Router eingesetzt werden.

Auch die Middleware für Client und Server bedarf noch weiterer Entwicklungen. Ziel sollten hierbei integrierte Lösungen sein, die die entsprechenden Dienste/Anwendungen einbinden. Zwar können mit den derzeitigen Implementierungen alle Funktionen des kontextsensitiven Routings realisiert werden, für einen produktiven Einsatz reicht dies aber noch nicht aus. Im Speziellen müsste für den Client untersucht werden, wie eine Kommunikation zwischen herkömmlichen kontextsensitiven Anwendungen und dem erweiterten AODV erfolgen kann. Einen vielversprechenden Ansatz hierzu liefert [Renh08]. Als aussichtsreich erscheint dabei die Entwicklung und Bereitstellung einer API. Über diese könnte dann jede Anwendung angepasst werden. Aktuell wurde zur besseren Handhabbarkeit und effizienten Verwaltung des Kontextrouters mit Arbeiten an einer grafischen Oberfläche begonnen. Auch für die Serververwaltung soll zukünftig ein *Graphical User Interface* (GUI) folgen. Weitere detaillierte Vorschläge für zukünftige Arbeiten an Clients und Server wurden bereits in den Abschnitten 8.3.2 und 8.4.2 diskutiert.

Die Routingprotokolle sind einer stetigen Weiterentwicklung ausgesetzt. Deshalb ist bei zukünftigen Arbeiten darauf zu achten, welche dieser Protokollentwicklungen in Bezug auf das kontextsensitive Routing attraktiv wären und ggf. sogar Vorteile gegenüber dem AODV besitzen. Es muss dann jeweils entschieden werden, ob die Routerarchitektur dem „neuen“ Routingprotokoll anzupassen ist.

Schließlich ist die Architektur auch auf Sicherheitsrisiken hin zu untersuchen. Bei den in Click realisierten Kontextrouter wird außerdem eine Überführung in den *Kernel*-Modus empfohlen.

11. Zusammenfassung

Die vorliegende Arbeit war von den Herausforderungen mobiler Ad-hoc-Netze motiviert. Durch deren dynamische Topologie können bisher aus Infrastrukturnetzen bekannte Verfahren zur Nutzung und zum Verwalten von Netzwerken nicht einfach übernommen werden. Dies gilt insbesondere für die Routingverfahren und für die Suche nach Diensten sowie deren Bereitstellung. Aus dieser Erkenntnis heraus wurde nach einer Möglichkeit gesucht, um den Nutzern eines solchen Netzwerkes den Zugriff auf Dienste zu ermöglichen. Der Trend zu kontextsensitiven Diensten war dabei zu berücksichtigen.

Bereits in der Einleitung (Kapitel 1) wurde darauf hingewiesen, dass zur Lösung dieser Aufgabe eine Zusammenführung von Routing und Dienstsuche als sehr günstig erschien, weil so ein für den Nutzer auf ihn am besten passender Dienstanbieter gefunden werden kann. Darüber hinaus müssen zur Bereitstellung von Diensten zwar netzübergreifende Lösungen geschaffen werden, Ad-hoc-Netze stellen hierbei durch ihre dynamische Topologie aber eine besondere Herausforderung dar. Eine weitere Feststellung war, dass die Kommunikation in heterogenen Netzwerken auch zukünftig vorrangig über das IP erfolgen wird. Zur Schaffung einer breiten Wissensgrundlage für die Konzeption eines kontextsensitiven Routingverfahrens wurden als erstes die Themengebiete Ad-hoc-Netze und Routing untersucht. Grundlage für deren Auswahl war dabei die Erkenntnis, dass in Bezug auf das Routing Ad-hoc-Netze gegenüber Infrastrukturnetzen anspruchsvoller sind.

Um die Anforderungen für ein solches Netz ermitteln zu können, ging Kapitel 2 zunächst auf die Eigenschaften von Ad-hoc-Netzen und die sich dadurch ergebenden Herausforderungen ein. Dabei konnte die Komplexität der Kommunikationsprozesse und die Besonderheiten gegenüber den Infrastrukturnetzen aufgezeigt werden. Ad-hoc-Netze bieten durch ihre Philosophie aber auch viele Vorteile. Sie sind selbstorganisierend und bedürfen keinerlei Infrastruktur. Ein Erfolg von Ad-hoc-Netzen wird hauptsächlich mit der Art ihrer Realisierung zusammenhängen. Allerdings werden sich Ad-hoc- und Infrastrukturnetze im Umfeld des *pervasive and ubiquitous Computing* sehr gut ergänzen. Am Ende des Kapitels wurden die aktuellen verfügbaren Technologien für Ad-hoc-Netze verglichen und gegenübergestellt. Im Ergebnis zeig-

te sich, dass aus heutiger Sicht WLANs am geeignetsten für die Realisierung von Ad-hoc-Netzen sind.

Anschließend erfolgte in Kapitel 3 eine Erläuterung der sich aus den Eigenschaften von Ad-hoc-Netzen ergebenden Anforderungen für Routingprotokolle und -verfahren. Es konnten die Herausforderungen und Probleme aufgezeigt werden, die sich aus dem dynamischen Charakter der Ad-hoc-Netze ergeben. Handover nehmen dabei einen zentralen Stellenwert ein. Die sich aus den beschriebenen Eigenschaften ergebenden unbedingten und optionalen Anforderungen an Routingprotokolle wurden erläutert. Weiterhin erfolgte ein Überblick über aktuelle Routingansätze. Deren Spektrum umfasst reaktive, proaktive, hybride und auf spezielle Anforderungen (positionsbasierte, energiesparende, usw.) hin entwickelte Protokolle. Die dabei genutzten Funktionsprinzipien wurden vorgestellt. Die „speziellen“ Routingprotokolle erwiesen sich für den in dieser Arbeit verfolgten Ansatz als nicht geeignet. Allgemein ergab sich aus den Erkenntnissen, dass zur Umsetzung des kontextsensitiven Routings nur ein solches Ad-hoc-Netz-Protokoll als Basis dienen kann, das ein möglichst breites Anwendungsspektrum unterstützt und trotzdem den Anforderungen des kontextsensitiven Routings genügt.

Kapitel 4 ging deshalb auf die Thematik der kontextsensitiven Diensterbringung ein. Für die Begriffe Kontext, Kontexttyp und Kontextinformation wurden Definitionen vorgestellt und erläutert. Dabei konnte festgestellt werden, dass eine vollständige Beschreibung des Kontextes weder möglich noch nötig ist. Darüber hinaus erfolgte eine Erläuterung der notwendigen Voraussetzungen für eine kontextsensitive Diensterbringung wie Kontexterfassung und -verarbeitung. Der Begriff Dienst wurde für diese Arbeit neu definiert und dabei um das Attribut „kontextsensitiv“ erweitert. Des Weiteren fanden Untersuchungen zu aktuellen Methoden der Dienstbereitstellung statt. Dadurch konnten deren Einsatzmöglichkeiten für das kontextsensitive Routing bewertet werden. Im Ergebnis stellte sich heraus, dass die vorgestellten Verfahren kontextsensitive Dienste nicht oder nur unzureichend unterstützen. Zudem hat der Netzprovider keine Möglichkeiten auf die Kommunikationsvorgänge einzuwirken (z. B. für Monitoring, Lastverteilung). Es konnte auch gezeigt und begründet werden, dass durch die Trennung von Dienstsuche und Diensterbringung die Flexibilität des Verfahrens erhöht wird. Eine weitere wichtige Erkenntnis war, dass eine Dienstsuche auf Netzwerkebene effizienter als auf Anwendungsebene ist.

Das Kapitel 5 baute auf die in den vorangegangenen Kapiteln gesammelten Erkenntnisse auf und stellte zwei verschiedene Ansätze für die Suche nach kontextsensitiven Diensten gegenüber. Der Anycast-Ansatz erwies sich bei der Möglichkeit Dienste zu adressieren und auszuwählen als nicht sehr flexibel. Der Ansatz einer zentralen Datenbank bot dagegen keine ausreichende Verfügbarkeit. Diese Ergebnisse verbunden mit den Erkenntnissen aus Kapitel 4 führten zu einem Architekturvorschlag, bei dem bestimmte Netzknoten – die Kontextrouter – über Kontextwissen verfügen. Er vereinigt die Vorteile der beiden ursprünglichen Ansätze und zeichnet sich durch seine Robustheit, Skalierbarkeit sowie Effizienz aus. Der neue Architekturvorschlag unterstützt heterogene Netzwerke und ermöglicht eine unabhängige flexible Dienstadressierung. Der Einsatz von Kontextroutern bietet die Möglichkeit, zukünftig Lastverteilungsmechanismen zu integrieren. Der Ansatz ist durch die Trennung von Dienstsuche und Diensterbringung interoperabel zu den herkömmlichen Verfahren und erfüllt dennoch alle an eine zukünftige Architektur gestellten Anforderungen.

Dem Architekturvorschlag folgend, wurde der Begriff des kontextsensitiven Routings erläutert und definiert.

Diese Ergebnisse dienten nun in Kapitel 6 als Basis für ein detailliertes Konzept zur Umsetzung der Architektur. Zu Beginn erfolgte dazu eine Erläuterung sämtlicher Kommunikationsvorgänge. Daraus ergaben sich zwei Möglichkeiten, die Dienstkommunikation im Anschluss an die Dienstsuche zu realisieren. Die Proxykommunikation bot dabei mehr Vorteile gegenüber einer direkten Kommunikation und entsprach besser den Anforderungen an das Konzept. Mit den Ergebnissen aus der Untersuchung der Routingprotokolle in Kapitel 3 und den sich aus Kapitel 5 ergebenden Anforderungen erwies sich AODV am geeignetsten als Basis für das zu entwickelnde Routingprotokoll. Daraufhin wurde dessen Verhalten in heterogenen Netzwerkumgebungen diskutiert und nachgewiesen, dass die Anforderungen des Konzeptes erfüllt werden. In der anschließenden konzeptionellen Diskussion entstanden als Ergebnis Lösungen zu den einzelnen Kommunikationsfunktionen (z. B. Adressierung, Dienstanmeldung, Dienstanfrage, Kommunikation zwischen Client und Server etc.). Dabei flossen die Schlussfolgerungen der vorhergehenden Kapitel als Anforderungen mit in das Konzept ein. Sofern mehrere Lösungen für die Realisierung bestimmter Funktionen bereitstanden, wurden diese verglichen und die für das Konzept geeignetste ausgewählt (z. B. bei der Adressierung). Alle dafür getroffenen Entscheidungen wurden begründet. Es erfolgten detaillierte Beschreibungen zu den notwendigen Modifikationen an den beteiligten Protokollen (AODV, ICMP, IP). Besonderer Wert lag bei der Entwicklung der Protokollerweiterungen auf Standardkonformität. Auch die Aufgaben der an der Routingarchitektur beteiligten Netzknoten wurden ausführlich diskutiert und beschrieben. Darüber hinaus erfolgten Vorschläge zu optionalen Funktionen (z. B. Auswahl alternativer Server, Möglichkeit von Blacklists). Diesbezüglich konnten auch drei Auswahlalgorithmen entwickelt und vorgestellt werden, mit denen über den Dienst und die beim Client zur Verfügung stehenden Kontexttypen ein für den Nutzer geeigneter Server gefunden wird. Der aktuellen Entwicklung geschuldet, gestalten sich alle Lösungen für das Architekturkonzept so, dass dieses auf IPv6 portierbar ist. Sofern bei den einzelnen Protokollen diesbezüglich Modifikationen notwendig sind, erfolgte an den entsprechenden Stellen eine Erläuterung. Eine Gegenüberstellung mit den Verfahren zur Dienstsuche aus Abschnitt 4.6 beweist die Überlegenheit des vorgestellten Konzeptes. Zusammengefasst werden durch das Konzept folgende Eigenschaften unterstützt:

- Umsetzung des kontextsensitiven Routings
- Dienstsuche unabhängig von einer IP-Adresse
- Unterstützung von Ad-hoc-Netzen
- Unterstützung heterogener IP-Netzwerkumgebungen
- Interoperabilität bzw. Kompatibilität zu herkömmlichen Netzwerkprotokollen und Verfahren zur Dienstsuche
- Einsatzmöglichkeit unter IPv6
- Eingriffsmöglichkeit für Provider

- Unabhängigkeit von der Diensterbringung
- hohe Robustheit

In Kapitel 7 wurde zunächst die Notwendigkeit von Simulationen erläutert. Hierbei erfolgte eine kritische Diskussion darüber, wo deren Einsatz zweckmäßig ist und in welcher Beziehung dazu Bewegungsmodelle stehen. Als Simulator erwies sich der ns-2 wegen seiner Quelloffenheit, dem Funktionsumfang und der Unterstützung verschiedener Netzarten als am geeignetsten. Dieser musste entsprechend dem Konzept angepasst werden. Erste Arbeiten dazu ermöglichten eine Integration kontextsensitiver Knoten in die Simulationsumgebung und boten eine Lösung zur Adressierung dieser Knoten. Auf der Basis der hierbei durchgeführten Simulationen konnten Rückschlüsse für das Konzept gewonnen werden, welches somit wesentlich verbessert wurde. Da die Simulationen sich aber als sehr zeitaufwändig herausstellten und die Übertragbarkeit der Simulationsergebnisse in die Realität überprüft werden musste, wurde die Entscheidung getroffen, das Architekturkonzept im Rahmen einer praktischen Realisierung mit Hilfe eines Demonstrators zu verifizieren.

Eine Beschreibung der praktischen Realisierung aller zum Architekturkonzept gehörenden Funktionseinheiten erfolgte dann in Kapitel 8. Die einzelnen Komponenten – Kontextrouter, Client und Server – wurden hierbei als Middleware konzipiert, sodass sie flexibel in Netzwerken eingesetzt werden können. Das Kernelement bildet der Kontextrouter. Seine Funktionen wurden mit Hilfe der Routersoftware Click realisiert und ausführlich beschrieben. Dieses Werkzeug stellte sich für die Realisierung eines Routers im Vergleich mit anderen Möglichkeiten als am geeignetsten heraus. Die Beschreibung der Routerfunktionen erfolgte mit Hilfe eines Routergraphen. Auf Besonderheiten, Modifikationen und Erweiterungen bezüglich des Konzeptes wurde ausführlich hingewiesen. Mit der Software Click konnten alle vom Konzept geforderten Funktionen umgesetzt werden. Außerdem wurden Implementierungen entwickelt, die die Funktionen von Clients und Servern realisieren. Dazu existieren aktuell einzelne Funktionseinheiten, die für einen produktiven Einsatz noch weiterentwickelt werden. Insgesamt bieten die in diesem Kapitel vorgestellten Implementierungen jedoch eine sehr gute Grundlage für ein Demonstratornetzwerk.

Kapitel 9 zeigte den erfolgreichen Nachweis der bei den Implementierungen realisierten zum kontextsensitiven Routing notwendigen Funktionen. Dazu wurde ein Demonstrator aufgebaut, der sowohl Ad-hoc- als auch Infrastrukturnetze unterstützt. Für die jeweiligen im Konzept geforderten Funktionen fanden entsprechende Tests statt. Dabei wurde die Funktionstüchtigkeit sowohl in Ad-hoc-Netzen als auch in heterogenen Netzwerkumgebungen nachgewiesen. Auch die Interoperabiltät zum unmodifizierten AODV konnte bestätigt werden. Das in Kapitel 6 vorgeschlagene Architekturkonzept ist somit verifiziert. Alle Messungen wurden detailliert beschrieben und anschließend ausgewertet, sodass die Ergebnisse reproduzierbar sind. Ziel des durchgeführten Perfomancetests war es, erste Aussagen über die Leistungsfähigkeit des optionalen Reroutings zu ermöglichen. Die Ergebnisse sind jedoch noch durch die Funktionsweise des AODV beeinflusst. Hier gibt es derzeit noch Entwicklungsbedarf und Potential für zukünftige Forschungen. Vorschläge für Verbesserungsmöglichkeiten wurden unterbreitet. Insgesamt stellt der Demonstrator eine solide Basis für zukünftige Tests und Entwicklungen dar und ist gerade auch mit Unterstützung der

optionalen Funktionen ein flexibles Mittel zum Test von realen kontextsensitiven Diensten.

Abgeschlossen wird die vorliegende Arbeit durch einen Ausblick, in dem Richtlinien für zukünftige Forschungs- und Entwicklungsarbeiten an der kontextsensitiven Routingarchitektur und deren Implementierungen vorgeschlagen werden. Diese bauen auf dem beschriebenen funktionstüchtigen kontextsensitiven Routing auf. Das ursprüngliche Konzept bleibt dabei erhalten.

Zusammenfassend erfüllt das vorgestellte Architekturkonzept und der daraus entwickelte Demonstrator somit alle gestellten Anforderungen und Erwartungen aus Abschnitt 1.2. Die Realisierung des Demonstrators und die Ergebnisse der Tests sind mit Hilfe der genannten Quellen und der Anhänge reproduzierbar.

A. Routerparameter

Die folgenden Tabellen A.1 und A.2 zeigen einen Auszug von ausgewählten Parametern, mit deren Hilfe bestimmte Kenngrößen im Demonstrator gezielt verändert werden können. Verwaltet werden diese Parameter in der Datei `context_config.hh`. Sie werden sowohl vom Kontextrouter als auch von den über Click realisierten Testfunktionen genutzt. Die aufgelisteten Werte waren während der Funktionstests und Messungen als Default gesetzt.

Parameter	Wert	Beschreibung
`CONTEXT_MAX_SERVICES`	100	Unterstützung von 100 verschiedenen Diensten
`CONTEXT_MAX_CTYPES`	100	Unterstützung von 100 verschiedenen Kontexttypen
`CONTEXT_REFRESH_INTERVAL`	5000	Intervall zwischen zwei gesendeten erweiterten RREQ der Clientanwendung aus Abschnitt 8.3.1
`CONTEXT_SERVERFILES_MAX_AGE`	6000	max. Gültigkeit der Registrierungsdateien (600 s im regulären Betrieb)
`CONTEXT_SERVERFILES_PATH`	`/home/register`	Verzeichnis für die Registrierungsdateien
`CONTEXT_SERVERFILES_EXTENSION`	`.list`	Endung für Registrierungsdateien
`CONTEXT_SERVERFILES_EOL`	;	Begrenzung der zu einem Dienst gehörenden Informationen
`CONTEXT_SERVERFILES_SHUTDOWN`	shutdown	Nachricht zur Anzeige des Herunterfahrens eines Servers
`CONTEXT_PRIO_WEIGHT`	2	j = 2 bei Algorithmus zur gewichteten Auswahl
`CONTEXT_RREQ_XTYPE`	16	Feld *Type* in der Erweiterung des RREQ
`CONTEXT_RREP_XTYPE`	17	Feld *Type* in der Erweiterung des RREP
`CONTEXT_CLIENTTABLE_MAX_AGE`	15	Gültigkeitsdauer für ungenutzte Einträge der Clienttabelle
`CONTEXT_REROUTE_RULE`	0	Rerouting zu Servern mit um `CONTEXT_REROUTE_MAX_SCOREDIFF` von X_k unterschiedlichen (0) oder gleichem (1) Werten (nur bei dynamischer Rangliste)
`CONTEXT_REROUTE_MAX_SCOREDIFF`	50	erlaubte prozentuale Abweichung von X_k (nur bei dynamischer Rangliste

Tabelle A.1: Auszug der wichtigsten Parameter Teil 1

Parameter	Wert	Beschreibung
CONTEXT_MAX_FOUND_SERVERS	100	Anzahl zu verwaltender Server
ICMP_ADVERT_INIT_INTERVAL	16000	Intervall in ms beim Initiieren von Advertisements
ICMP_ADVERT_INIT_COUNT	3	Anzahl der nach Initiierung zu sendenden Advertisements
ICMP_ADVERT_MAX_INTERVAL	600000	max. Abstand zwischen zwei Advertisements in ms
ICMP_ADVERT_CODE	10	Feld *Code* im ICMP-Advertisement-Paket
ICMP_ADVERT_NUMADDRS	1	Anzahl der im ICMP-Advertisement-Paket übertragenen Routeradressen
ICMP_ADVERT_RESPONSE_DELAY	2000	max. Verzögerung in ms bis zur Beantwortung eines Solicitations
ICMP_SOLICIT_INTERVAL	10000	Intervall zwischen zwei Solicitations in ms
ICMP_SOLICIT_CODE	10	Feld *Code* im ICMP-Solicitation-Paket
ICMP_SOLICIT_RESERVED	0	Feld *Reserved* im ICMP-Solicitation-Paket

Tabelle A.2: Auszug der wichtigsten Parameter Teil 2

B. Routergraph

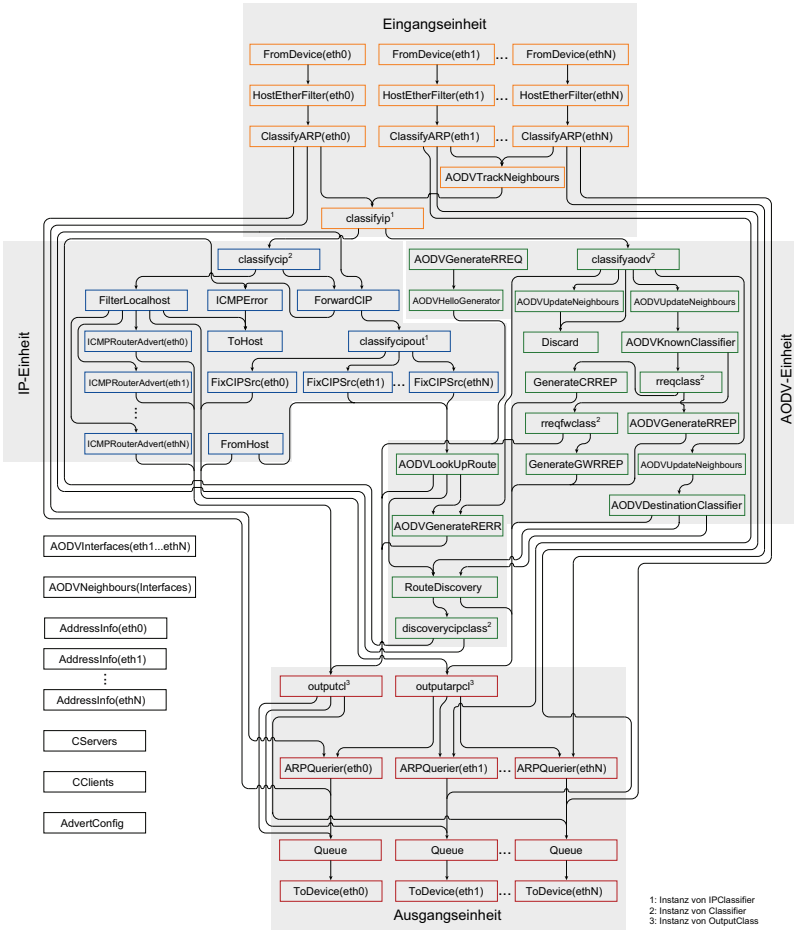


Abbildung B.1: Routergraph des Kontextrouters

C. Aktuelles Skript des Kontextrouters

Das folgende Skript basiert auf [Wenz07a] und wurde seitdem im Rahmen der vorliegenden Arbeit stetig weiterentwickelt. Die hier gelistete Version stammt vom März 2008.

```
//***************  configure interfaces   ***************
AddressInfo(mydev0 192.168.1.10/24 00:50:04:EE:96:52); //fixed network
AddressInfo(mydev1 192.168.2.10/24 00:50:04:EE:95:A8); //wired aodv network
AddressInfo(mydev2 10.0.0.10/24 00:02:2D:49:8C:FF); //wireless aodv network

interfaces::AODVInterfaces(eth1 192.168.2.10/24 00:50:04:EE:95:A8, eth3 10.0.0.10/24 00:02:2D:49:8C:FF);

//MAC filter configuration;
//for using MAC filter uncomment this region and region "with MAC filter"' and comment region "without MAC filter"
//macfilter::Classifier(<OFFSET>/<MAC_ADRESSE>, -);
//macfilter::Classifier(6/001D7D38EC93, -);

//*************** configure routingtables ***************
neighbours :: AODVNeighbours(interfaces);
cservers :: CServers;
clients :: CClients;
aconfig :: AdvertConfig;

//***************     element classes     ***************
elementclass OutputClass{
  input[0]
    -> ipsrc :: IPClassifier(src mydev0, src mydev1, src mydev2, -);
  ipsrc[0]
    -> [0]output;
  ipsrc[1]
    -> [1]output;
  ipsrc[2]
    -> [2]output;
  ipsrc[3]
    -> ipdst :: IPClassifier(dst net mydev0, dst net mydev1, dst net mydev2, -);
  ipdst[0]
    -> [0]output;
  ipdst[1]
    -> [1]output;
  ipdst[2]
    -> [2]output;
  ipdst[3]
    -> tee::Tee(3);
  tee[0]
    -> [0]output;
  tee[1]
    -> [1]output;
  tee[2]
    -> [2]output;
}

elementclass OutputEth0{
  input[0]
    -> Queue(2000)
    -> ToDevice(eth0);
}

elementclass OutputEth1{
  input[0]
    -> Queue(2000)
    -> ToDevice(eth1);
}

elementclass OutputEth3{
  input[0]
    -> Queue(2000)
    -> todev::ToDevice(eth3);
}

elementclass InputEth0{
  $myaddr_ethernet |
  FromDevice(eth0)
    -> HostEtherFilter($myaddr_ethernet, DROP_OWN false, DROP_OTHER true)
    -> output;
}

elementclass InputEth1{
```

```
  $myaddr_ethernet |
  FromDevice(eth1)
    -> HostEtherFilter($myaddr_ethernet, DROP_OWN false, DROP_OTHER true)
    -> Paint(0)
    -> output;
}

//without MAC filter;
//for using MAC filter comment this region, set configuration parameters above and uncomment region "with MAC filter"
elementclass InputEth3{
  $myaddr_ethernet |
  FromDevice(eth3)
    -> HostEtherFilter($myaddr_ethernet, DROP_OWN false, DROP_OTHER true)
    -> Paint(1)
    -> output;
}

//with MAC filter
/*
elementclass InputEth3{
  $myaddr_ethernet |
  FromDevice(eth3)
    -> HostEtherFilter($myaddr_ethernet, DROP_OWN false, DROP_OTHER true)
    -> macfilter::Classifier(6/00E000D1D034, -);
  macfilter[0]
    -> Discard;
  macfilter[1]
    -> Paint(1)
    -> output;
}
*/
//end of MAC filter

elementclass ClassifyARP{
  $myaddr |
  input[0]
    -> arpclass :: Classifier(12/0806 20/0001, 12/0806 20/0002, -); //arp request, arp reply, other
  arpclass[0]
    -> ARPResponder($myaddr, $myaddr)
    -> [0]output;
  arpclass[1]
    -> [1]output
  arpclass[2]
    -> CheckIPHeader(OFFSET 14)
    -> MarkIPHeader(14)
    -> [2]output;
}

elementclass System{
  FromHost(fake0, mydev0)
    -> fromhost_cl :: Classifier(12/0806, -); //arp request, other
  FromHost(fake1, mydev1)
    -> fromhost_cl;
  FromHost(fake2, mydev2)
    -> fromhost_cl;
  fromhost_cl[0]
    -> ARPResponder(0.0.0.0/0 1:1:1:1:1:1)
    -> ToHost(fake0);
  fromhost_cl[1]
    -> Strip(14)
    -> CheckIPHeader
    -> MarkIPHeader
    -> output_cl :: IPClassifier(dst net mydev0, -);
  output_cl[0]
    -> [0]output;
  output_cl[1]
    -> [1]output;
  input[0]
    -> dstclass :: IPClassifier(dst net mydev0, dst net mydev1, dst net mydev2);
  dstclass[0]
    -> EtherEncap(0x0800, 1:1:1:1:1:1, mydev0)
    -> CheckIPHeader(14)
    -> MarkIPHeader(14)
    -> ToHost(fake0);
  dstclass[1]
    -> EtherEncap(0x0800, 1:1:1:1:1:1, mydev1)
    -> CheckIPHeader(14)
    -> MarkIPHeader(14)
    -> ToHost(fake1);
  dstclass[2]
    -> EtherEncap(0x0800, 1:1:1:1:1:1, mydev2)
    -> CheckIPHeader(14)
    -> MarkIPHeader(14)
    -> ToHost(fake2);
}

elementclass FilterLocalhost{
  input
//ping responder; if not desired comment region and uncomment region "without ping responder"
    -> localhost :: IPClassifier(dst host mydev0, dst host mydev1, dst host mydev2, ip proto icmp && icmp type routersolicit, - );
//cip0, cip1, cip2, solicit, other
  localhost[0]
    -> ping :: ICMPPingResponder;
  localhost[1]
    -> ping;
  localhost[2]
    -> ping;
  localhost[3]
    -> Strip(14)
    -> classifyicmpcode :: Classifier(21/0a, -) //icmp-code = 10
```

```
//  -> classifyadvertdst :: IPClassifier(dst net mydev0, dst net mydev1, dst net mydev2); //Broadcast x.x.x.255
    -> classifyadvertdst :: Classifier(12/C0A801, 12/C0A802, 12/0A0000); //Broadcast 255.255.255.255
  localhost[4]
    -> classifyipdst :: IPClassifier(dst net mydev0, -);
  ping[0]
    -> Strip(14)
    -> CheckICMPHeader
    -> CheckIPHeader
    -> MarkIPHeader
    -> classifyipdst;
  ping[1]
    -> [0]output;
//without ping responder
/*
  -> localhost :: IPClassifier(dst host mydev0, dst host mydev1, dst host mydev2, ip proto icmp && icmp type routersolicit, - );
  //cip0, cip1, cip2, solicit, other
  localhost[0]
    -> [0]output;
  localhost[1]
    -> [0]output;
  localhost[2]
    -> [0]output;
  localhost[3]
    -> Strip(14)
    -> classifyicmpcode :: Classifier(21/0a, -) //icmp-code = 10
    -> classifyadvertdst :: IPClassifier(dst net mydev0, dst net mydev1, dst net mydev2);
  localhost[4]
    -> classifyipdst :: IPClassifier(dst net mydev0, -);
*/
//end of ping
  classifyicmpcode[1]
    -> Discard;
  classifyadvertdst[0]
    -> ICMPRouterAdvert(mydev0, 0, aconfig)
//  -> IPEncap(1, mydev0, 192.168.1.255, TTL 2) //Broadcast 192.168.1.255
    -> IPEncap(1, mydev0, 255.255.255.255, TTL 2) //Broadcast 255.255.255.255
    -> EtherEncap(0x0800, mydev0, ff:ff:ff:ff:ff:ff)
    -> [1]output;
  classifyadvertdst[1]
    -> ICMPRouterAdvert(mydev1, 0, aconfig)
//  -> IPEncap(1, mydev1, 192.168.2.255, TTL 2) //Broadcast 192.168.2.255
    -> IPEncap(1, mydev1, 255.255.255.255, TTL 2)//Broadcast 255.255.255.255
    -> EtherEncap(0x0800, mydev1, ff:ff:ff:ff:ff:ff)
    -> [1]output;
  classifyadvertdst[2]
    -> ICMPRouterAdvert(mydev2, 0, aconfig)
//  -> IPEncap(1, mydev2, 10.0.0.255, TTL 2) //Broadcast 10.0.0.255
    -> IPEncap(1, mydev2, 255.255.255.255, TTL 2)//Broadcast 255.255.255.255
    -> EtherEncap(0x0800, mydev2, ff:ff:ff:ff:ff:ff)
    -> [1]output;
  classifyipdst[0]
    -> [2]output;
  classifyipdst[1]
    -> [3]output;
}

elementclass RouteDiscovery{
  $genrreq |
  input[0]
    -> [0]discovery :: AODVWaitingForDiscovery($genrreq,neighbours); //data
  input[1]
    -> [1]discovery; //rrep
  discovery[0]
    -> [0]output; //found
  discovery[1]  //not found
    -> [1]output;
  AODVLinkNeighboursDiscovery(neighbours,discovery);
}

//***************  element declarations   ***************
outputcl :: OutputClass;
outputarpcl :: OutputClass;
outputeth0 :: OutputEth0;
outputeth1 :: OutputEth1;
outputeth3 :: OutputEth3;
classifyarpeth0 :: ClassifyARP(mydev0);
classifyarpeth1 :: ClassifyARP(mydev1);
classifyarpeth3 :: ClassifyARP(mydev2);
arpquerierеth0 :: ARPQuerier(mydev0);
arpquerierеth1 :: ARPQuerier(mydev1);
arpquerierеth3 :: ARPQuerier(mydev2);
classifyip :: IPClassifier(dst udp port 654, -);  //AODV, other
classifyaodv :: Classifier(42/01, 42/03, 42/02 22/01, 42/02, - ); //RREQ, RERR, HELLO, RREP, other
classifycip :: Classifier(34/9E, -); //cip (on=30, cf=1; 34.Byte/ in Hex)
cipforwarder :: ForwardCIP(cservers, clients);
system :: System;
classifycipout :: IPClassifier(dst net mydev0, dst net mydev1, dst net mydev2, -);
filterlocalhost :: FilterLocalhost;
lookup :: AODVLookUpRoute(neighbours);
discoverycipclass :: Classifier(20/9E, -); //cip (on=16, cf=1), without ethernet-header
rreqclass :: Classifier(66/10, - ); //crreq (extension-type=4)
destinationclassifier :: AODVDestinationClassifier(neighbours);
setrrepheaders::AODVSetRREPHeaders()
routereply :: AODVGenerateRREP(neighbours,setrrepheaders);
rreqfwclass :: Classifier(50/C0A801, -); //destination net = 192.168.1.x
rerr:: AODVGenerateRERR(neighbours)
  -> outputcl;
knownclassifier :: AODVKnownClassifier(neighbours);
hello::AODVHelloGenerator(neighbours)
  -> outputcl;
```

```
genrreq :: AODVGenerateRREQ(neighbours,knownclassifier)
  -> hello;
routediscovery :: RouteDiscovery(genrreq);

//***************   element connections   ***************
InputEth0(mydev0)
  -> classifyarpeth0;
classifyarpeth0[0]
  -> outputeth0; //arp request
classifyarpeth0[1]
  -> [1]arpquerierеth0; //arp reply
classifyarpeth0[2]
  -> classifyip; //other

InputEth1(mydev1)
  -> classifyarpeth1;
classifyarpeth1[0]
  -> outputeth1; //arp request
classifyarpeth1[1]
  -> [1]arpquerierеth1; //arp reply
classifyarpeth1[2]
  -> AODVTrackNeighbours(rerr, neighbours)
  -> classifyip; //other

InputEth3(mydev2)
  -> classifyarpeth3;
classifyarpeth3[0]
  -> outputeth3; //arp request
classifyarpeth3[1]
  -> [1]arpquerierеth3; //arp reply
classifyarpeth3[2]
  -> AODVTrackNeighbours(rerr, neighbours)
  -> classifyip; //other

classifyip[0]
  -> classifyaodv;
classifyip[1]
  -> classifycip;

classifycip[0] //cip
  -> Strip(14)
  -> CheckIPHeader
  -> [0]cipforwarder
  -> CheckIPHeader
  -> MarkIPHeader
  -> classifycipout;
classifycip[1] //ip
  -> filterlocalhost;

icmperr::ICMPError(mydev1, 3, 1)
  -> system;

cipforwarder[1]
  ->  icmperr;

classifycipout[0]
  -> FixCIPSrc(mydev0)
  -> outputarpcl;
classifycipout[1]
  -> FixCIPSrc(mydev1)
  -> Paint(3)
  -> lookup;
classifycipout[2]
  -> FixCIPSrc(mydev2)
  -> Paint(3)
  -> lookup;
classifycipout[3]
  -> Paint(3)
  -> lookup;

system[0]
  -> outputarpcl
system[1]
  -> Paint(3)
  -> lookup;

filterlocalhost[0] //ip for system
  -> Strip(14)
  -> system;
filterlocalhost[1] //advert
  -> outputcl;
filterlocalhost[2] //ip for others
  -> Strip(14)
  -> outputarpcl;
filterlocalhost[3] //ip for others
  -> Paint(3)
  -> lookup;

lookup[0] //known
  -> StripToNetworkHeader
  -> outputarpcl;
lookup[1] //unknown
  -> [0]routediscovery;
lookup[2] //rerr
  -> rerr;

routediscovery[0] //found
  -> SetIPChecksum
  -> StripToNetworkHeader
  -> outputarpcl;
```

```
routediscovery[1] //not found
  -> discoverycipclass;

discoverycipclass[0] //cip
  -> [1]cipforwarder
discoverycipclass[1] //ip
  -> icmperr;

classifyaodv[0] //RREQ
  -> AODVUpdateNeighbours(neighbours)
  -> knownclassifier;
classifyaodv[1] //RERR
  -> [1]rerr;
classifyaodv[2] //HELLO
  -> AODVUpdateNeighbours(neighbours)
  -> Discard;
classifyaodv[3] //RREP
  -> AODVUpdateNeighbours(neighbours)
  -> destinationclassifier;
classifyaodv[4] //other
  -> Discard;

knownclassifier[0] //reply
  -> rreqclass;
knownclassifier[1] //forward
  -> rreqfwclass;

rreqclass[0] //CRREQ
  -> GenerateCRREP(cservers,clients,neighbours,setrrepheaders)
  -> setrrepheaders;
rreqclass[1] //CRREP
  -> routereply;

destinationclassifier[0]
  -> [1]routediscovery;
destinationclassifier[1]
  -> SetIPChecksum
  -> SetUDPChecksum
  -> StripToNetworkHeader
  -> outputarpcl;
destinationclassifier[2]
  -> Paint(2)
  -> [0]routediscovery;

rreqfwclass[0] //dst in net mydev0 -> reply
  -> GenerateGWRREP(neighbours,setrrepheaders)
  -> setrrepheaders;
rreqfwclass[1] //dst unknown -> forward
  -> outputcl;

routereply
  -> setrrepheaders
  -> SetUDPChecksum
  -> SetIPChecksum
  -> outputarpcl;

outputcl[0]
  -> outputeth0;
outputcl[1]
  -> outputeth1;
outputcl[2]
  -> outputeth3;

outputarpcl[0]
  -> arpqueriereth0
  -> outputeth0;
outputarpcl[1]
  -> arpqueriereth1
  -> outputeth1;
outputarpcl[2]
  -> arpqueriereth3
  -> outputeth3;
```

D. Elemente

Die Tabellen D.1 und D.2 listen die Elemente des im Anhang B dargestellten Routergraphen auf und identifizieren die Herkunft des zu den Elementen gehörenden Quellcodes – für „AODV“ ist dies [Brae05] und für „Click“ [CMRP07]. Solche Elemente, deren Quellcode im Rahmen der vorliegenden Arbeit modifiziert oder neu entwickelt wurde, sind in der Spalte „Status“ entsprechend (mit „neu“/“modifiziert“) gekennzeichnet.

Bei *ClassifyARP(ethi)*, *FilterLocalhost* und *RouteDiscovery* handelt es sich um Elementklassen, die sich aus weiteren Elementen zusammensetzen. Die Strukturen und Elemente basieren auf [Brae05] und wurden ggf. modifiziert. Die Elementklassen selbst sind im Skript des Kontextrouters in Anhang C dargestellt.

Der Quellcode modifizierter AODV-Elemente kann aus Gründen des Copyrights nicht veröffentlicht werden, liegt aber unter [Debe08] vor. Für neue Elemente ist der Quellcode in Anhang G aufgeführt, sofern dieser nicht bereits in [Wenz07a] bereitgestellt oder seitdem weiterentwickelt wurde. Die Angaben zum Quellcode dienen der Reproduzierbarkeit der Ergebnisse der vorliegenden Arbeit.

Element	Ursprung	Status	Quellcode
AddressInfo(ethi)	Click	original	[CMRP07]
AdvertConfig	neu	neu	Anhang G
AODVDestinationClassifier	AODV	modifiziert	[Debe08]
AODVGenerateRERR	AODV	modifiziert	[Debe08]
AODVGenerateRREP	AODV	modifiziert	[Debe08]
AODVGenerateRREQ	AODV	modifiziert	[Debe08]
AODVHelloGenerator	AODV	modifiziert	[Debe08]
AODVInterfaces(eth1 ... ethN)	neu	neu	Anhang G
AODVKnownClassifier	AODV	modifiziert	[Debe08]
AODVLookUpRoute	AODV	modifiziert	[Debe08]
AODVNeighbours(Interfaces)	AODV	modifiziert	[Debe08]
AODVTrackNeighbours	AODV	modifiziert	[Debe08]
AODVUpdateNeighbours	AODV	modifiziert	[Debe08]
ARPQuerier(ethi)	Click	original	[CMRP07]
CClients	neu	neu	Anhang G
Classifier	Click	original	[CMRP07]
CServers	neu	neu	Anhang G
Discard	Click	original	[CMRP07]
FixCIPSrc(ethi)	Click	modifiziert	[Wenz07a]
ForwardCIP	neu	neu	Anhang G

Tabelle D.1: Die Elemente des Routergraphs Teil 1

Element	Ursprung	Status	Quellcode
FromDevice(ethi)	Click	original	[CMRP07]
FromHost	Click	original	[CMRP07]
GenerateCRREP	neu	neu	Anhang G
GenerateGWRREP	neu	neu	[Wenz07a]
HostEtherFilter(ethi)	Click	original	[CMRP07]
ICMPRouterAdvert(ethi)	neu	neu	[Wenz07a]
ICMPError	Click	original	[CMRP07]
IPClassifier	Click	original	[CMRP07]
OutputClass	Click	original	[CMRP07]
Queue	Click	original	[CMRP07]
ToDevice(ethi)	Click	original	[CMRP07]
ToHost	Click	original	[CMRP07]

Tabelle D.2: Die Elemente des Routergraphs Teil 2

E. Konfiguration der Netzknoten

Die folgenden Tabellen E.1 und E.2 enthalten Informationen zur Hardwareausstattung und Konfiguration der bei den Tests und Messungen verwendeten Netzknoten.

Knoten	1	
Prozessor	Intel Pentium III 450 MHz	
Speicher (RAM)	192 MByte	
Betriebssystem	Linux (Gentoo)	
Kernel	2.6.23-gentoo-r3	
Click	1.6.0	
Netzwerk	3Com 3C900B-COMBO	Netgear WG311v2
Netzwerkstandard	IEEE 802.3a/i	IEEE 802.11b/g
IP-Adresse	192.168.1.20	10.0.0.20
MAC-Adresse	00:10:5a:39:52:fc	00:09:5b:bb:36:08
Knoten	2	3
Prozessor	Mobile Intel Pentium IV 2,2 GHz	Intel Celeron MMX 300 MHz
Speicher (RAM)	768 MByte	192 MByte
Betriebssystem	Linux (Gentoo)	Linux (Gentoo)
Kernel	2.6.23-gentoo-r3	2.6.23-gentoo-r3
Click	1.6.0	1.6.0
Netzwerk	Intersil Corporation Prism 2.5	D-Link DE-528
Netzwerkstandard	IEEE 802.11b	IEEE 802.3a/i
IP-Adresse	10.0.0.30	192.168.1.30
MAC-Adresse	00:e0:00:d1:d0:34	00:05:5d:d3:d5:ec

Tabelle E.1: Die Konfiguration der Client-/Serverrechner

Knoten	Kontextrouter		
Prozessor	Intel Pentium III 800 MHz		
Speicher (RAM)	256 MByte		
Betriebssystem	Linux (Gentoo)		
Kernel	2.6.22-gentoo-r9		
Click	1.6.0		
Netzwerk	3Com Corporation 3C905B-TX		Dell Drue Mobile 1150 Series PC Card
Netzwerkstandard	IEEE 802.3i/u		IEEE 802.11b
IP-Adresse	192.168.1.10	192.168.2.10	10.0.0.10
MAC-Adresse	00:50:04:ee:96:52	00:50:04:ee:95:a8	00:02:2d:49:8c:ff

Tabelle E.2: Die Konfiguration des Kontextrouters

F. Messreihen

Während des in Abschnitt 9.4 beschriebenen Performancetests wurden die in den Tabellen F.1 und F.2 dargestellten Messreihen aufgenommen. Sie umfassen 60 Messungen aus zwei Messzyklen, die mit Hilfe von Wireshark ermittelt worden sind.

Nr.	Beginn des Reroutings in s	Ende des Reroutings in s	Anfrage zum alternativen Server in s	Antwort vom alternativen Server in s	Dauer des Reroutings in s
1	55,299172	63,798604	54,021442	54,023255	8,497619
2	246,288847	254,785825	245,010783	245,012639	8,495122
3	742,764217	751,278733	741,500124	741,502001	8,512639
4	895,327043	903,824280	894,047594	894,049369	8,495462
5	1334,528082	1342,384342	1332,610649	1332,612485	7,854424
6	1666,201821	1673,981551	1664,282803	1664,284600	7,777933
7	1772,907584	1780,597213	1771,023618	1771,025438	7,687809
8	1874,657394	1883,044086	1873,403402	1873,405297	8,384797
9	1957,469736	1965,139144	1955,564205	1955,566099	7,667514
10	2045,005345	2052,684849	2043,109991	2043,111840	7,677655
11	2126,223898	2134,523815	2124,948371	2124,950292	8,297996
12	2248,437082	2256,136629	2246,562374	2246,564229	7,697692
13	2334,206562	2342,515247	2332,940732	2332,942556	8,306861
14	2532,021832	2539,697952	2530,121803	2530,123686	7,674237
15	2715,908147	2724,217729	2714,642680	2714,644532	8,307730
16	2900,566098	2908,956599	2899,319764	2899,321562	8,388703
17	2989,388583	2997,092860	2987,500124	2987,501975	7,702426
18	3075,989846	3084,302452	3074,727358	3074,729144	8,310820
19	3157,780460	3166,108986	3156,532532	3156,534303	8,326755
20	3240,643498	3248,315758	3238,740536	3238,742301	7,670495
21	3324,988706	3333,290847	3323,716172	3323,718033	8,300280
22	3478,052169	3486,359743	3476,785775	3476,787600	8,305749
23	3559,238600	3566,948707	3557,373719	3557,375559	7,708267
24	3642,374649	3650,683713	3641,108681	3641,110450	8,307295
25	3730,241829	3738,552292	3728,977341	3728,979099	8,308705
26	3815,261400	3823,572767	3813,996880	3813,998664	8,309583
27	3896,433912	3904,129633	3894,548582	3894,550443	7,693860
28	3979,465039	3987,900059	3978,197860	3978,199712	8,433168
29	4112,833459	4121,142311	4111,567783	4111,569609	8,307026
30	4211,000641	4219,309168	4209,734749	4209,736587	8,306689

Tabelle F.1: Messreihe 1

Die Spalte „Nr.“ gibt die betreffende Messung an. „Beginn des Reroutings“ ist der Zeitstempel für das an der Schnittstelle zum Ad-hoc-Netz eintreffende erste Paket, das nach Ausfall des ursprünglichen Servers nicht mehr weitergeleitet werden kann.

Nr.	Beginn des Reroutings in s	Ende des Reroutings in s	Anfrage zum alternativen Server in s	Antwort vom alternativen Server in s	Dauer des Reroutings in s
31	283,827381	292,234377	287,710233	287,712084	8,405145
32	420,518767	428,227451	423,771776	423,773548	7,706912
33	503,447326	511,746350	507,292270	507,294083	8,297211
34	592,126631	600,426080	595,971936	595,973820	8,297565
35	696,442046	704,150954	699,694543	699,696332	7,707119
36	777,228456	785,535017	781,081070	781,082904	8,304727
37	862,221817	870,520106	866,066239	866,068085	8,296443
38	944,249929	952,563952	948,107967	948,109799	8,312191
39	1027,022397	1035,384140	1030,928101	1030,929871	8,359973
40	1110,439025	1118,741234	1114,286449	1114,288229	8,300429
41	1194,169780	1202,476263	1198,022577	1198,024385	8,304675
42	1275,477713	1283,774835	1279,320731	1279,322624	8,295229
43	1357,002105	1365,303128	1360,848745	1360,850527	8,299241
44	1442,438958	1450,745757	1446,291966	1446,293756	8,305009
45	1524,368351	1532,665877	1528,212564	1528,214371	8,295719
46	1608,260945	1616,560803	1612,106643	1612,108433	8,298068
47	1757,717314	1766,117737	1761,569275	1761,571112	8,398586
48	1841,510881	1849,818600	1845,363831	1845,365599	8,305951
49	1925,217497	1933,526394	1929,071264	1929,073083	8,307078
50	2010,382381	2018,085973	2013,631957	2013,633811	7,701738
51	2097,632355	2105,297657	2100,831225	2100,833885	7,662642
52	2272,414110	2280,147072	2275,633757	2275,635680	7,731039
53	2357,958211	2366,272969	2361,819134	2361,821007	8,312885
54	2441,115270	2449,424740	2444,969706	2444,971544	8,307632
55	2525,653887	2534,028583	2529,510272	2529,512142	8,372826
56	2610,058380	2617,717927	2613,264220	2613,266133	7,657634
57	2694,455271	2702,759772	2698,306149	2698,307957	8,302693
58	2784,132515	2792,447452	2787,992348	2787,994210	8,313075
59	2877,010210	2885,324838	2880,870278	2880,872107	8,312799
60	2961,031050	2969,356618	2964,902886	2964,904707	8,323747

Tabelle F.2: Messreihe 2

„Ende des Reroutings“ ist der Zeitstempel des ersten erfolgreich durch Rerouting an der Schnittstelle zum Ad-hoc-Netz übergebenen Antwortpaketes zur Weiterleitung an den Client. Die dritte und die vierte Spalte zeigen die Zeitstempel für das Absenden des Ping-Paketes vom Kontextrouter zum alternativen Server und für den Empfang der zugehörigen Antwort. Die Zeitstempel der beiden Netzwerkschnittstellen unterscheiden sich deswegen, weil die Messungen mit jeweils einer Wireshark-Instanz je Schnittstelle durchgeführt wurden. Die Uhr startet bei Wireshark mit Aufruf der Instanz, sodass sich die absoluten Werte der Zeitstempel zwischen den Netzwerkschnittstellen unterscheiden. Eine Zuordnung kann über die relativen Zeitabstände zweier Kommunikationsvorgänge erfolgen. Schließlich gibt die letzte Spalte die Zeit wieder, die allein für den Reroutingprozess aufgewendet wurde. Hier ist bereits die Zeitdauer, die der Server für eine Antwort benötigt (Spalten vier und fünf), abgezogen worden.

Abbildung F.1 zeigt das zusammengefasste Ergebnis der Messungen. Durchschnittlich wurden zum Rerouting 8,16 s benötigt. Die Gründe dafür wurden bereits in Abschnitt 9.4 erläutert. Darüber hinaus ist aus den Ergebnissen in den Tabellen zu schließen, dass der Einfluss der Antwortzeit durch den alternativen Server vernachlässigbar ist. Die Schwankung des Zeitwertes für das Rerouting ergibt sich aus den Netzwerkeigenschaften wie beispielsweise dem Medienzugriff oder der Routenfindung im Ad-hoc-Netz. So ist dort mit größer werdender Struktur und höherer Dynamik in der Topologie auch mit einer größeren Schwankungsbreite zu rechnen.

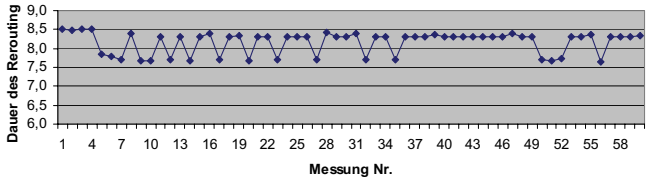


Abbildung F.1: Dauer des Rerouting

G. Programmcode

Im Folgenden wird der Quellcode der Elemente aufgeführt, die nicht in [Wenz07a] veröffentlicht bzw. seitdem modifiziert oder aber neu entwickelt wurden (siehe Anhang D). Ein funktionsfähiger Kontextrouter kann somit unter zusätzlicher Verwendung von [Debe08] rekonstruiert werden. Alle Tests sind reproduzierbar. Ein Element setzt sich jeweils aus einer Header-Datei (<Datei>.hh) und einer Quelltext-Datei (<Datei>.cc) zusammen. Die Programmierung und Modifikation selbst erfolgte durch eine wissenschaftliche Hilfskraft im Rahmen dieser Arbeit (siehe [Debe08]).

AdvertConfig

context_advertconfig.hh

```
#ifndef ADVERTCONFIG_HH
#define ADVERTCONFIG_HH
#include <click/element.hh>

CLICK_DECLS

class AdvertConfig : public Element {
    public:

AdvertConfig();
~AdvertConfig();

const char *class_name() const { return "AdvertConfig"; }
const char *port_count() const  { return "0"; }
const char *processing() const { return "h"; }

int configure(Vector<String> &, ErrorHandler *);
        int initialize(ErrorHandler *);

        void add_handlers();

        int _max_interval;

    private:

        static int write_handler(const String&, Element*, void*, ErrorHandler*);

};

CLICK_ENDDECLS
#endif
```

context_advertconfig.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include "context_advertconfig.hh"

CLICK_DECLS
AdvertConfig::AdvertConfig()
{
}

AdvertConfig::~AdvertConfig()
{
}

int
AdvertConfig::configure(Vector<String> &conf, ErrorHandler *errh){
  if (cp_va_parse(conf, this, errh, cpEnd) < 0)
    return -1;
  _max_interval = 600000;
  return 0;
}

int
AdvertConfig::initialize(ErrorHandler *){
```

```
  return 0;
}

enum { H_INTERVAL };

int
AdvertConfig::write_handler(const String &str_in, Element *e, void *thunk, ErrorHandler *errh) {
  String s = cp_uncomment(str_in);
  AdvertConfig *adv = static_cast<AdvertConfig *>(e);
  if ((uintptr_t)thunk == H_INTERVAL) {
    if (!cp_integer(s, (int *)&adv->_max_interval))
      return errh->error("'interval' should be an integer");
    if (adv->_max_interval < 0) {
      adv->_max_interval = 600;
      return errh->error("'interval' should be greater than 0, setting back to 600000", adv->_max_interval);
    }
    click_chatter("ICMPAdvert: maximum advertisement interval changed to %dms", adv->_max_interval);
  }
  return 0;
}

void
AdvertConfig::add_handlers()
{
  add_write_handler("interval", write_handler, (void *)H_INTERVAL);
}

CLICK_ENDDECLS

EXPORT_ELEMENT(AdvertConfig AdvertConfig-Context_AdvertConfig)
```

AODVInterfaces(eth1 . . . ethN)

aodv_interfaces.hh

```
#ifndef AODVINTERFACES_HH
#define AODVINTERFACES_HH
#include <click/element.hh>
#include <click/hashmap.hh>
#include "click_aodv.hh"

CLICK_DECLS

struct iftable_entry{
  IPAddress    ipaddr;
  IPAddress    ipmask;
  EtherAddress ethaddr;
  String       devname;
};
typedef HashMap<uint8_t,iftable_entry> InterfaceMap;

class AODVInterfaces : public Element {
    public:

AODVInterfaces();
~AODVInterfaces();

const char *class_name() const { return "AODVInterfaces"; }
const char *port_count() const  { return "0"; }
const char *processing() const { return "h"; }

int configure(Vector<String> &, ErrorHandler *);
        int initialize(ErrorHandler *);

        void insertInterfaces();
        iftable_entry findIFbyID(uint8_t);
        uint8_t findIFbyMAC(EtherAddress);
        uint8_t getNumIF();

    private:
        InterfaceMap interfaces;
        iftable_entry _entry[AODV_MAX_IF];
};

CLICK_ENDDECLS
#endif
```

aodv_interfaces.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>

#include "aodv_interfaces.hh"

CLICK_DECLS
AODVInterfaces::AODVInterfaces()
{
}
```

```
AODVInterfaces::~AODVInterfaces()
{
}

int
AODVInterfaces::configure(Vector<String> &conf, ErrorHandler *errh){
  if (conf.size() == 0) {
    errh->error("please specify at least one interface");
    return -1;
  }
  if (conf.size() > AODV_MAX_IF) {
    errh->error("to many interfaces specified");
    return -1;
  }
  for (int i = 0; i < conf.size(); i++) {
    Vector<String> parts;
    cp_spacevec(conf[i], parts);
    if (parts.size() != 3){
      errh->error("expected 'NAME IPADDR/PREFIX ETHADDR', got '%s'", conf[i].c_str());
      return -1;
    }
    if (cp_string(parts[0], &_entry[i].devname) &&
        cp_ip_prefix(parts[1], &_entry[i].ipaddr, &_entry[i].ipmask, false) &&
        cp_ethernet_address(parts[2], &_entry[i].ethaddr)){
        //click_chatter("%i %s %s %s %s", i, _entry[i].devname.c_str(), _entry[i].ipaddr.s().c_str(), _entry[i].ipmask.s().c_str(), \
        _entry[i].ethaddr.s().c_str());
    }
    else {
      errh->error("expected 'NAME IPADDR/PREFIX ETHADDR', got '%s'", conf[i].c_str());
    }
  }
  //insert every information in routing table
  insertInterfaces();
  return 0;
}

int
AODVInterfaces::initialize(ErrorHandler *){
//   iftable_entry temp;
//   temp = findIFbyID(1);
//   click_chatter("%s", temp.devname.c_str());
//   uint8_t i;
//   i = getNumIF();
//   click_chatter("%i", i);

  return 0;
}

void
AODVInterfaces::insertInterfaces(){
  uint8_t i = 0;
  while (_entry[i].devname != "") {
    interfaces.insert(i,_entry[i]);
    //click_chatter("AODVInterfaces: inserted %s %s %s %s", _entry[i].devname.c_str(), _entry[i].ipaddr.s().c_str(), \
    _entry[i].ipmask.s().c_str(), _entry[i].ethaddr.s().c_str());
    i++;
  }
}

iftable_entry
AODVInterfaces::findIFbyID(uint8_t id){
  //return table entry by id
  InterfaceMap::Pair* read = interfaces.find_pair(id);
  iftable_entry buffer;
  if (read) {
    buffer.ipaddr = read->value.ipaddr;
    buffer.ipmask = read->value.ipmask;
    buffer.ethaddr = read->value.ethaddr;
    buffer.devname = read->value.devname;
  }
  return buffer;
}

uint8_t
AODVInterfaces::getNumIF(){
  //return number of interfaces
  uint8_t i = 0;
  for(InterfaceMap::const_iterator iter = interfaces.begin(); iter != interfaces.end(); ++iter)
    i++;
  return i;
}

uint8_t
AODVInterfaces::findIFbyMAC(EtherAddress eth){
  //return interface id by mac address
  uint8_t id = 0;
  uint8_t i = 0;
  for(InterfaceMap::const_iterator iter = interfaces.begin(); iter != interfaces.end(); ++iter){
    if (iter.value().ethaddr == eth)
      id = i;
    i++;
  }
  return id;
}

//macro magic
#include <click/bighashmap.cc>
#if EXPLICIT_TEMPLATE_INSTANCES
template class HashMap<uint8_t,iftable_entry>;
#endif
```

```
CLICK_ENDDECLS

EXPORT_ELEMENT(AODVInterfaces AODVInterfaces-AODV_Interfaces)
```

CClients

context_cclients.hh

```
#ifndef CCLIENTS_HH
#define CCLIENTS_HH
#include <click/element.hh>
#include <click/hashmap.hh>

#include "context_config.hh"

CLICK_DECLS

typedef HashMap<IPAddress, struct clienttable> ClientMap;

class CClients : public Element {
    public:

        CClients();
        ~CClients();

        const char *class_name() const  { return "CClients"; }
        const char *port_count() const  { return "-"; }
        const char *processing() const  { return "a"; }

        int configure(Vector<String> &, ErrorHandler *);

        void insertClientEntry(struct FindClientCRREQ);
        void deleteoldClientEntries();
        uint16_t determineSessionID(uint32_t);
        ResultClientCRREQ findClientEntry(uint32_t, uint16_t);
        void deleteClientEntry(uint32_t, uint16_t);
        void updateClientEntry(uint32_t, uint32_t, uint32_t);
        ResultClientOSCT findOSCT(uint32_t, uint32_t, uint16_t);
        void updateTimestamp(uint32_t, uint16_t);
        ServerList findNextServer(FindNxtServer);
    private:
        ClientMap clients;
};

CLICK_ENDDECLS
#endif
```

context_cclients.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>

#include "context_cclients.hh"

CLICK_DECLS
CClients::CClients()
{
}

CClients::~CClients()
{
}

int
CClients::configure(Vector<String> &conf, ErrorHandler *errh)
{
    if (cp_va_parse(conf, this, errh, cpEnd) < 0) return -1;
    return 0;
}

void
CClients::insertClientEntry(FindClientCRREQ data){
  deleteoldClientEntries();
  //if data.id is not in table, insert entry
  if (!clients.find_pair(data.id)) {
    clienttable buffer;
    buffer.id = data.id;
    buffer.ca[0].st = data.st;
    buffer.ca[0].sid = data.sid;
    buffer.ca[0].ts = data.ts;
    buffer.ca[0].ls = data.ls;
    buffer.ca[0].os = data.ls;
    memcpy(buffer.ca[0].osct, data.osct, sizeof(data.osct));
    memcpy(buffer.ca[0].ct, data.ct, sizeof(data.ct));
    memcpy(buffer.ca[0].sl, data.sl, sizeof(data.sl));
    clients.insert(buffer.id,buffer);
    click_chatter("CClients: inserted service %u at %s with server %s", data.st, IPAddress(data.id).s().c_str(), \
    IPAddress(data.ls).s().c_str());
  }
  //if data.id is in table, look for other conditions
```

```
  else {
    //search in table
    for(ClientMap::const_iterator iter = clients.begin(); iter != clients.end(); ++iter){
      if (iter.value().id == data.id) {
        int i = 0;
        bool stpair = false;
        bool ctpair = false;
        int stpos = 0;
        do {
          //click_chatter("id %u st %u", iter.value().id, iter.value().ca[i]);
          if (iter.value().ca[i].st == data.st) {
            stpair = true;
            //look, if any data.ct is different from table entry
            for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
              //left side of uint8_t
              if ((iter.value().ca[i].ct[j] >> 4) != (data.ct[j] >> 4)) {
                ctpair = true;
                stpos = i;
              }
              //right side of uint8_t
              if ((iter.value().ca[i].ct[j] & 0xF) != (data.ct[j] & 0xF)) {
                ctpair = true;
                stpos = i;
              }
            }
            //if session id has changed, update entry
            if (iter.value().ca[i].sid != data.sid) {
              ctpair = true;
              stpos = i;
            }
          }
          i++;
        } while (iter.value().ca[i].st != 0);
        clienttable buffer;
        //insert entry with data.st, if not in table
        if (stpair == false) {
          buffer.id = iter.value().id;
          memcpy(buffer.ca, iter.value().ca, sizeof(iter.value().ca));
          buffer.ca[i].st = data.st;
          buffer.ca[i].sid = data.sid;
          buffer.ca[i].ts = data.ts;
          buffer.ca[i].ls = data.ls;
          buffer.ca[i].os = data.ls;
          memcpy(buffer.ca[i].osct, data.osct, sizeof(data.osct));
          memcpy(buffer.ca[i].ct, data.ct, sizeof(data.ct));
          memcpy(buffer.ca[i].sl, iter.value().ca[i].sl, sizeof(iter.value().ca[i].sl));
          clients.remove(buffer.id);
          clients.insert(buffer.id,buffer);
          click_chatter("CClients: %d append st %u at entry %s with server %s",i, data.st, \
          IPAddress(data.id).s().c_str(),IPAddress(data.ls).s().c_str());
        }
        else {
          //update ct in entry, if data.ct changed
          if (ctpair == true) {
            buffer.id = iter.value().id;
            memcpy(buffer.ca, iter.value().ca, sizeof(iter.value().ca));
            buffer.ca[stpos].st = data.st;
            buffer.ca[stpos].sid = data.sid;
            buffer.ca[stpos].ts = data.ts;
            buffer.ca[stpos].ls = data.ls;
            buffer.ca[stpos].os = data.ls;
            memcpy(buffer.ca[stpos].osct, data.osct, sizeof(data.osct));
            memcpy(buffer.ca[stpos].ct, data.ct, sizeof(data.ct));
            memcpy(buffer.ca[stpos].sl, iter.value().ca[stpos].sl, sizeof(iter.value().ca[stpos].sl));
            clients.remove(buffer.id);
            clients.insert(buffer.id,buffer);
            click_chatter("CClients: updated service %u at entry %s with sid %u and server %s", data.st, \
            IPAddress(data.id).s().c_str(), data.sid, IPAddress(data.ls).s().c_str());
          }
        }
      }
    }
  }
}

void
CClients::updateTimestamp(uint32_t id_in, uint16_t sid_in) {
  ClientMap::Pair* read = clients.find_pair(id_in);
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      if (read->value.ca[i].sid == sid_in) {
        clienttable buffer;
        buffer.id = read->value.id;
        memcpy(buffer.ca, read->value.ca, sizeof(read->value.ca));
        clients.remove(buffer.id);
        Timestamp now = Timestamp::now();
        buffer.ca[i].ts = now.sec();
        clients.insert(buffer.id, buffer);
      }
      i++;
    }
  }
}

void
CClients::deleteoldClientEntries(){
  //search in every table entry
  ClientCRREQ buffer[CONTEXT_MAX_SERVICES];
  for(ClientMap::const_iterator iter = clients.begin(); iter != clients.end(); ++iter){
```

```
    int i = 0;
    int j = 0;
    bool oldentries = false;
    //click_chatter("deine mudder 1 %u %u", iter.value().id, iter.value().ca[i].ts);
    while (iter.value().ca[i].st != 0) {
      //look, if there's an old entry
      Timestamp now = Timestamp::now();
      uint32_t ts = now.sec()-CONTEXT_CLIENTTABLE_MAX_AGE;
      if (iter.value().ca[i].ts > ts) {
        buffer[j] = iter.value().ca[i];
        j++;
      }
      else {
        oldentries = true;
      }
      i++;
    }
    if (oldentries == true) {
      clients.remove(iter.value().id);
      //if no st is left, delete whole entry
      if (buffer[0].st != 0) {
        clienttable entry;
        entry.id = iter.value().id;
        memcpy(entry.ca, buffer, sizeof(buffer));
        clients.insert(entry.id, entry);
      }
    }
  }
}

uint16_t
CClients::determineSessionID(uint32_t id) {
  ClientMap::Pair* read = clients.find_pair(id);
  int sid = 0;
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      if (read->value.ca[i].sid > sid) {
        sid = read->value.ca[i].sid;
        updateTimestamp(id, sid);
        deleteoldClientEntries();
      }
      i++;
    }
  }
  sid++;
  //click_chatter("sid %u at %u",sid, id);
  return sid;
}

ResultClientOSCT
CClients::findOSCT(uint32_t src_in, uint32_t os_in, uint16_t st_in) {
  ResultClientOSCT result;
  result.f = false;
  memset(result.osct, '\0', sizeof(result.osct));
  ClientMap::Pair* read = clients.find_pair(src_in);
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      if ((read->value.ca[i].os == os_in) && (read->value.ca[i].st == st_in)) {
        memcpy(result.osct, read->value.ca[i].osct, sizeof(read->value.ca[i].os));
        result.f = true;
        updateTimestamp(src_in, read->value.ca[i].sid);
        deleteoldClientEntries();
      }
      i++;
    }
  }
  return result;
}

ResultClientCRREQ
CClients::findClientEntry(uint32_t src_in, uint16_t sid_in) {
  updateTimestamp(src_in, sid_in);
  deleteoldClientEntries();
  ResultClientCRREQ result;
  ClientMap::Pair* read = clients.find_pair(src_in);
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      if (read->value.ca[i].sid == sid_in) {
        click_chatter("CClients: found entry with id %s and sid %u", IPAddress(src_in).s().c_str(), sid_in);
        result.id = read->value.id;
        result.nr = read->value.nr;
        result.ls = read->value.ca[i].ls;
        result.os = read->value.ca[i].os;
        result.st = read->value.ca[i].st;
        memcpy(result.ct, read->value.ca[i].ct, sizeof(read->value.ca[i].ct));
      }
      else {
        result.id = 0;
      }
      i++;
    }
  }
  else {
    result.id = 0;
  }
  return result;
}
```

```
void
CClients::deleteClientEntry(uint32_t id_in, uint16_t sid_in){
  //search in every table entry
  ClientCRREQ buffer[CONTEXT_MAX_SERVICES];
  ClientMap::Pair* read = clients.find_pair(id_in);
  if (read) {
    int i = 0;
    int j = 0;
    while (read->value.ca[i].st != 0) {
      if (read->value.ca[i].sid != sid_in) {
        click_chatter("CClients: delete entry at id %s with sid %u", IPAddress(read->value.id).s().c_str(), read->value.ca[i].sid);
        buffer[j] = read->value.ca[i];
        j++;
      }
      i++;
    }
    clients.remove(read->value.id);
    //if no st is left, delete whole entry
    if (buffer[0].st != 0) {
      clienttable entry;
      entry.id = read->value.id;
      memcpy(entry.ca, buffer, sizeof(buffer));
      clients.insert(entry.id, entry);
    }
  }
}

void
CClients::updateClientEntry(uint32_t id_in, uint32_t ls_in, uint32_t ns_in) {
  ClientCRREQ buffer[CONTEXT_MAX_SERVICES];
  ClientMap::Pair* read = clients.find_pair(id_in);
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      buffer[i] = read->value.ca[i];
      if (read->value.ca[i].ls == ls_in) {
        click_chatter("CClients: update - delete ls %s, insert ns %s at entry %s", IPAddress(buffer[i].ls).s().c_str(), \
        IPAddress(ns_in).s().c_str(), IPAddress(id_in).s().c_str());
        buffer[i].ls = ns_in;
        updateTimestamp(id_in, buffer[i].sid);
        deleteoldClientEntries();
      }
      i++;
    }
    clients.remove(read->value.id);
    clienttable entry;
    entry.id = read->value.id;
    memcpy(entry.ca, buffer, sizeof(buffer));
    clients.insert(entry.id, entry);
  }
}

ServerList
CClients::findNextServer(FindNxtServer data){
  ServerList result;
  result.id = 0;
  for (int i = 0; i>CONTEXT_MAX_CTYPES; i++) {
    result.ct[i] = 0;
  }
  result.na = 1;
  ClientMap::Pair* read = clients.find_pair(data.client);
  if (read) {
    int i = 0;
    while (read->value.ca[i].st != 0) {
      int j = 0;
      while (read->value.ca[i].sl[j].id != 0) {
        //click_chatter("server %s %d", IPAddress(read->value.ca[i].sl[j].id).s().c_str(), read->value.ca[i].sl[j].id);
        if (read->value.ca[i].sl[j].id == data.server) {
          result.id = read->value.ca[i].sl[j+1].id;
          memcpy(result.ct, read->value.ca[i].sl[j+1].ct, sizeof(read->value.ca[i].sl[j+1].ct));
          if (read->value.ca[i].sl[j+2].id != 0)
            result.na = 0;
        }
        j++;
      }
      i++;
    }
  }
  return result;
}

//macro magic
#include <click/bighashmap.cc>
#if EXPLICIT_TEMPLATE_INSTANCES
template class HashMap<IPPair, void *>;
template class HashMap<IPAddress, int>;
#endif

CLICK_ENDDECLS

EXPORT_ELEMENT(CClients CClients-Context_CClients)
```

CServers

context_cservers.hh

```
#ifndef CSERVERS_HH
#define CSERVERS_HH
#include <click/element.hh>
#include <click/hashmap.hh>

#include "context_config.hh"

CLICK_DECLS

typedef HashMap<IPAddress, ctable> ContextMap;

class CServers : public Element {
    public:

        CServers();
        ~CServers();

        const char *class_name() const  { return "CServers"; }
        const char *port_count() const  { return "-"; }
        const char *processing() const  { return "a"; }

        int configure(Vector<String> &, ErrorHandler *);
        void add_handlers();

        CRREQResult findRerouteContextEntry(CRREQFind);
        CRREQResult findContextEntry(CRREQFind);
        CRREQResult findContextEntryMethod0(CRREQFind);
        CRREQResult findContextEntryMethod1(CRREQFind);
        CRREQResult findContextEntryMethod2(CRREQFind);
        CRREQResult findContextEntryMethod3(CRREQFind);
        void addContextEntry();
        void readContextEntry(uint32_t);
        void deleteoldContextEntries();
        void deleteContextFile(uint32_t);
        void deleteContextEntry(uint32_t);
        void updateContextEntry(struct ctable);
        void insertContextEntry(struct ctable);
        void readContextEntryByST(uint16_t);
        void updateContextTable();

        uint32_t _fmethod;
    private:
        static int write_handler(const String&, Element*, void*, ErrorHandler*);

        ContextMap contexts;
};

CLICK_ENDDECLS
#endif
```

context_cservers.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include <dirent.h>
#include <sys/stat.h>

#include "context_cservers.hh"

CLICK_DECLS
CServers::CServers()
{
}

CServers::~CServers()
{
}

int
CServers::configure(Vector<String> &conf, ErrorHandler *errh)
{
  if (cp_va_parse(conf, this, errh, cpEnd) < 0) return -1;
  _fmethod = 1;
  return 0;
}

CRREQResult
CServers::findRerouteContextEntry(CRREQFind data){
  CRREQResult result;
  float serverprio, clientprio, score, bufferscore, idfoundscore;
  bool matchingct[CONTEXT_MAX_CTYPES];
  bool idfound = false;
  int entrycount = 0;
  result.id = 0;
  result.st = 0;
  result.na = true;
  result.sid = 0;
  for (int i=0; i<CONTEXT_MAX_CTYPES; i++) {
    result.ct[i] = 0;
```

```
}
if (data.id != 0) {
  memcpy(result.ct, data.osct, sizeof(data.osct));
}
else {
  memset(result.ct, 0, sizeof(result.ct));
}
score = 0;
bufferscore = 0;
idfoundscore = 0;

//first: search, if data.id is in table
if (data.id != 0) {
  ContextMap::Pair* read = contexts.find_pair(data.id);
  if (read) {
    int i = 0;
    //search in servicetypes
    while (read->value.ca[i].st != 0) {
      //if servicetype fits, search for contexts
      if (read->value.ca[i].st == data.st) {
        serverprio = 0;
        clientprio = 0;
        memset(matchingct, false, sizeof(matchingct));
        //search for fitting contexts and calculate score
        for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
          //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
          //left
          if ((data.ct[j] >> 7) == 1)
            clientprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
          //right
          if (((data.ct[j] & 0xF) >> 3) == 1)
            clientprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);

          //if iter.contexttype[x]=1 -> serverprio+=prio(contexttype[x])^j
          //even
          if (read->value.ca[i].ct[j*2] == 1)
            serverprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
          //odd
          if (read->value.ca[i].ct[(j*2)+1] == 1)
            serverprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);
        }
        idfound = true;
        idfoundscore = ((serverprio/clientprio) * 100) - CONTEXT_REROUTE_MAX_SCOREDIFF;
      }
    i++;
    }
  }
}

//search in table
for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
  int i = 0;
  //search in servicetypes
  while (iter.value().ca[i].st != 0) {
    //if servicetype fits, search for contexts
    if (iter.value().ca[i].st == data.st) {
      serverprio = 0;
      clientprio = 0;
      memset(matchingct, false, sizeof(matchingct));

      //search for fitting contexts and calculate score
      for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
        //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
        //left
        if ((data.ct[j] >> 7) == 1)
          clientprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
        //right
        if (((data.ct[j] & 0xF) >> 3) == 1)
          clientprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);

        //if iter.contexttype[x]=1 -> serverprio+=prio(contexttype[x])^j
        //even
        if (iter.value().ca[i].ct[j*2] == 1) {
          serverprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
          matchingct[j*2] = true;
        }
        //odd
        if (iter.value().ca[i].ct[(j*2)+1] == 1) {
          serverprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);
          matchingct[(j*2)+1] = true;
        }
      }
      //if score is higher than score from last entry and data.id was not found
      //choose server for result
      if ((score < (serverprio/clientprio) * 100) && (idfound == false)) {
        bufferscore = score;
        score = (serverprio/clientprio) * 100;
        result.id = iter.value().id;
        result.st = iter.value().ca[i].st;
        memcpy(result.ct, iter.value().ca[i].ct, sizeof(iter.value().ca[i].ct));
        entrycount++;
      }
      else {
        //if data id was found, choose reroute rule
        if (CONTEXT_REROUTE_RULE) {
          //if all context types are the same like at data.id
          //and if the calculated score is > 0
          //choose server for result
          bool servermatches = true;
          for (int j = 0; j < CONTEXT_MAX_CTYPES; j++) {
```

```
            if ((matchingct[j]) && (result.ct[j] != iter.value().ca[i].ct[j])) {
              servermatches = false;
            }
          }
          if ((servermatches) && (((serverprio/clientprio) * 100) > 0))  {
            bufferscore = (serverprio/clientprio) * 100;
            score = (serverprio/clientprio) * 100;
            result.id = iter.value().id;
            result.st = iter.value().ca[i].st;
            memcpy(result.ct, iter.value().ca[i].ct, sizeof(iter.value().ca[i].ct));
            entrycount++;
          }
        }
        else {
          //if scorees: (last found server < found server < data.id)
          //choose server for result
          if ((idfoundscore > (serverprio/clientprio) * 100) && (bufferscore < (serverprio/clientprio) * 100)) {
            bufferscore = (serverprio/clientprio) * 100;
            score = (serverprio/clientprio) * 100;
            result.id = iter.value().id;
            result.st = iter.value().ca[i].st;
            memcpy(result.ct, iter.value().ca[i].ct, sizeof(iter.value().ca[i].ct));
            entrycount++;
          }
        }
      }
      //if only one entry is found, set na-flag
      if (entrycount > 1) {
        result.na = false;
click_chatter("na-flag is %d", result.na);
      }
    }
    i++;
  }
}
//if only the data.id server was found, return 0
if (result.id == data.id) {
  result.id = 0;
}
//if no server found, return only servicetype
if (result.id == 0) {
  result.st = data.st;
  click_chatter("cservers: no entry was found in the routingtable");
}
else {
  click_chatter("CServers: found best server with x_k %f and service %u at id %s na-flag is %d", score, result.st, \
  IPAddress(result.id).s().c_str(), result.na);
}
return result;
}

CRREQResult
CServers::findContextEntryMethod0(CRREQFind data){
  CRREQResult result;
  float serverprio, clientprio, score, idfoundscore;
  bool idfound = false;
  int entrycount = 0;
  result.id = 0;
  result.st = 0;
  result.na = true;
  result.sid = 0;
  for (int i=0; i<CONTEXT_MAX_CTYPES; i++) {
    result.ct[i] = 0;
  }

  score = -1;
  idfoundscore = -1;

  //first: search, if data.id is in table
  if (data.id != 0) {
    ContextMap::Pair* read = contexts.find_pair(data.id);
    if (read) {
      int i = 0;
      //search in servicetypes
      while (read->value.ca[i].st != 0) {
        //if servicetype fits, search for contexts
        if (read->value.ca[i].st == data.st) {
          serverprio = 0;
          clientprio = 0;
          //search for fitting contexts and calculate score
          for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
            //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
            //left
            if ((data.ct[j] >> 7) == 1)
              clientprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
            //right
            if (((data.ct[j] & 0xF) >> 3) == 1)
              clientprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);

            //if iter.contexttype[x]=1 -> serverprio+=prio(contexttype[x])^j
            //even
            if (read->value.ca[i].ct[j*2] == 1)
              serverprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
            //odd
            if (read->value.ca[i].ct[(j*2)+1] == 1)
              serverprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);
          }
          if (idfound == false) {
            idfound = true;
            idfoundscore = ((serverprio/clientprio) * 100);
```

```
        }
       }
      i++;
      }
    }
  }

  //search in table
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    int i = 0;
    //search in servicetypes
    while (iter.value().ca[i].st != 0) {
      //if servicetype fits, search for contexts
      if (iter.value().ca[i].st == data.st) {
        serverprio = 0;
        clientprio = 0;

        //search for fitting contexts and calculate score
        for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
          //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
          //left
          if ((data.ct[j] >> 7) == 1)
            clientprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
          //right
          if (((data.ct[j] & 0xF) >> 3) == 1)
            clientprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);

          //if iter.contexttype[x]=1 -> serverprio+=prio(contexttype[x])^j
          //even
          if (iter.value().ca[i].ct[j*2] == 1)
            serverprio += pow((data.ct[j] >> 4) & 0x7, CONTEXT_PRIO_WEIGHT);
          //odd
          if (iter.value().ca[i].ct[(j*2)+1] == 1)
            serverprio += pow((data.ct[j] & 0xF) & 0x7, CONTEXT_PRIO_WEIGHT);
        }
        //if score is higher than score from last entry and data.id was not found
        //choose server for result
        //click_chatter("score: %f, newscore: %f", score, (serverprio/clientprio)*100);
        if ((score < (serverprio/clientprio) * 100) && (idfound == false)) {
          score = (serverprio/clientprio) * 100;
          result.id = iter.value().id;
          result.st = iter.value().ca[i].st;
          memcpy(result.ct, iter.value().ca[i].ct, sizeof(iter.value().ca[i].ct));
        }
        else {
          //if scorees: (last found server < found server < data.id)
          //choose server for result
          if ((idfoundscore > (serverprio/clientprio) * 100) && (score < (serverprio/clientprio) * 100)) {
            score = (serverprio/clientprio) * 100;
            result.id = iter.value().id;
            result.st = iter.value().ca[i].st;
            memcpy(result.ct, iter.value().ca[i].ct, sizeof(iter.value().ca[i].ct));
          }
        }
        if ((((serverprio/clientprio) * 100) > -1) && (iter.value().id != data.id)) {
      entrycount++;
        }
        //if only one entry is found, set na-flag
        if ((entrycount > 1) && (idfound == false)) {
          result.na = false;
  click_chatter("na-flag is now %d", result.na);
    }
      }
      i++;
    }
  }
  //if only the data.id server was found, return 0
  if (result.id == data.id) {
    result.id = 0;
  }
  //if no server found, return only servicetype
  if (result.id == 0) {
    result.st = data.st;
    click_chatter("cservers: no entry was found in the routingtable");
  }
  else {
    click_chatter("CServers: found best server with x_k %f and service %u at id %s na-flag is %d", score, result.st, \
    IPAddress(result.id).s().c_str(), result.na);
  }
  return result;
}

CRREQResult
CServers::findContextEntryMethod1(CRREQFind data){
  CRREQResult result;
  float serverprio, clientprio;
  ServerList serverlist[CONTEXT_MAX_FOUND_SERVERS];
  for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
    serverlist[i].id = 0;
    for (int j=0; j<CONTEXT_MAX_CTYPES; j++) {
      serverlist[i].ct[j] = 0;
    }
  }
  int fs = 0;

  //search in table
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    int i = 0;
    //search in servicetypes
    while (iter.value().ca[i].st != 0) {
```

```
      //if servicetype fits, search for contexts
      if (iter.value().ca[i].st == data.st) {
        serverprio = 0;
        clientprio = 0;
        //search for fitting contexts and calculate score
        for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
          //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
          if ((data.ct[j] >> 7) == 1) { //left
            clientprio += pow(((data.ct[j] >> 4) & 0x7) + 1, CONTEXT_PRIO_WEIGHT);
            if (iter.value().ca[i].ct[j*2] == 1)
              serverlist[fs].ct[j*2] = 1;
            if (iter.value().ca[i].ct[j*2] == 1) //even
              serverprio += pow(((data.ct[j] >> 4) & 0x7) + 1, CONTEXT_PRIO_WEIGHT);
          }
          if (((data.ct[j] & 0xF) >> 3) == 1) { //right
            clientprio += pow(((data.ct[j] & 0xF) & 0x7) + 1, CONTEXT_PRIO_WEIGHT);
            if (iter.value().ca[i].ct[(j*2)+1] == 1)
              serverlist[fs].ct[(j*2)+1] = 1;
            if (iter.value().ca[i].ct[(j*2)+1] == 1) //odd
              serverprio += pow(((data.ct[j] & 0xF) & 0x7) + 1, CONTEXT_PRIO_WEIGHT);
          }
        }
        serverlist[fs].id = iter.value().id;
        serverlist[fs].xk = (serverprio/clientprio) * 100;
        fs++;
      }
      i++;
    }
  }

  if (serverlist[0].id != 0) {
    //determine number of found servers
    int c = 0;
    while (serverlist[c].id != 0) {
      c++;
    }
    //sort that thing :)
    for(int m = 0; m < c; m++) {
      for(int n = 0; n < (c - 1); n++) {
        if (serverlist[n].xk < serverlist[n+1].xk) {
          ServerList temp;
          temp = serverlist[n];
          serverlist[n] = serverlist[n+1];
          serverlist[n+1] = temp;
        }
      }
    }
    //first server in list = result
    result.id = serverlist[0].id;
    result.sid = 0;
    result.st = data.st;
    if (c > 1)
      result.na = 0;
    else
      result.na = 1;
    memcpy(result.ct, serverlist[0].ct, sizeof(serverlist[0].ct));
    memcpy(result.sl, serverlist, sizeof(serverlist));
    click_chatter("CServers: found best server with x_k %f and service %u at id %s na-flag is %d", serverlist[0].xk, result.st, \
    IPAddress(result.id).s().c_str(), result.na);
  }
  else {
    result.id = 0;
    result.sid = 0;
    result.st = data.st;
    result.na = 1;
    click_chatter("na-flag is now %d", result.na);
    for (int i=0; i<CONTEXT_MAX_CTYPES; i++) {
      result.ct[i] = 0;
    }
    for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
      result.sl[i].id = 0;
    }
    click_chatter("CServers: no entry was found in the routingtable");
  }

  return result;
}

CRREQResult
CServers::findContextEntryMethod2(CRREQFind data){
  CRREQResult result;
  float serverprio, clientprio;
  ServerList serverlist[CONTEXT_MAX_FOUND_SERVERS];
  for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
    serverlist[i].id = 0;
    for (int j=0; j<CONTEXT_MAX_CTYPES; j++) {
      serverlist[i].ct[j] = 0;
    }
  }
  int fs = 0;

  //search in table
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    int i = 0;
    //search in servicetypes
    while (iter.value().ca[i].st != 0) {
      //if servicetype fits, search for contexts
      if (iter.value().ca[i].st == data.st) {
        serverprio = 0;
        clientprio = 0;
```

```
        //search for fitting contexts and calculate score
        for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
          //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
          if ((data.ct[j] >> 7) == 1) { //left
            clientprio += ((data.ct[j] >> 4) & 0x7) + 1;
            if (iter.value().ca[i].ct[j*2] == 1)
              serverlist[fs].ct[j*2] = 1;
            if (iter.value().ca[i].ct[j*2] == 1) //even
              serverprio += ((data.ct[j] >> 4) & 0x7) + 1;
          }
          if (((data.ct[j] & 0xF) >> 3) == 1) { //right
            clientprio += ((data.ct[j] & 0xF) & 0x7) + 1;
            if (iter.value().ca[i].ct[(j*2)+1] == 1)
              serverlist[fs].ct[(j*2)+1] = 1;
            if (iter.value().ca[i].ct[(j*2)+1] == 1) //odd
              serverprio += ((data.ct[j] & 0xF) & 0x7) + 1;
          }
        }
        serverlist[fs].id = iter.value().id;
        serverlist[fs].xk = (serverprio/clientprio) * 100;
        fs++;
      }
      i++;
    }
  }

  if (serverlist[0].id != 0) {
    //determine number of found servers
    int c = 0;
    while (serverlist[c].id != 0) {
      c++;
    }
    //sort that thing :)
    for(int m = 0; m < c; m++) {
      for(int n = 0; n < (c - 1); n++) {
        if (serverlist[n].xk < serverlist[n+1].xk) {
          ServerList temp;
          temp = serverlist[n];
          serverlist[n] = serverlist[n+1];
          serverlist[n+1] = temp;
        }
      }
    }
    //first server in list = result
    result.id = serverlist[0].id;
    result.sid = 0;
    result.st = data.st;
    if (c > 1)
      result.na = 0;
    else
      result.na = 1;
    memcpy(result.ct, serverlist[0].ct, sizeof(serverlist[0].ct));
    memcpy(result.sl, serverlist, sizeof(serverlist));
    click_chatter("CServers: found best server with x_k %f and service %u at id %s na-flag is %d", serverlist[0].xk, result.st, \
    IPAddress(result.id).s().c_str(), result.na);
  }
  else {
    result.id = 0;
    result.sid = 0;
    result.st = data.st;
    result.na = 1;
    click_chatter("na-flag is now %d", result.na);
    for (int i=0; i<CONTEXT_MAX_CTYPES; i++) {
      result.ct[i] = 0;
    }
    for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
      result.sl[i].id = 0;
    }
    click_chatter("CServers: no entry was found in the routingtable");
  }

  return result;
}

CRREQResult
CServers::findContextEntryMethod3(CRREQFind data){
  CRREQResult result;
  ServerList serverlist[CONTEXT_MAX_FOUND_SERVERS];
  for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
    serverlist[i].id = 0;
    for (int j=0; j<CONTEXT_MAX_CTYPES; j++) {
      serverlist[i].ct[j] = 0;
    }
  }
  int fs = 0;

  //search in table
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    int i = 0;
    //search in servicetypes
    while (iter.value().ca[i].st != 0) {
      //if servicetype fits, search for contexts
      if (iter.value().ca[i].st == data.st) {
        //search for fitting contexts and calculate score
        for (int j = 0; j < CONTEXT_MAX_CTYPES/2; j++) {
          //if data.contexttype[x]=1 -> clienttprio+=prio(contexttype[x])^j
          if (((data.ct[j] >> 7) == 1) && (iter.value().ca[i].ct[j*2] == 1)) { //left
              serverlist[fs].ct[j*2] = 1;
          }
          if ((((data.ct[j] & 0xF) >> 3) == 1) && (iter.value().ca[i].ct[(j*2)+1] == 1)) { //right
```

```
            serverlist[fs].ct[(j*2)+1] = 1;
          }
        }
        serverlist[fs].id = iter.value().id;
        serverlist[fs].xk = 0;
        fs++;
      }
      i++;
    }
  }

  if (serverlist[0].id != 0) {
    //determine number of found servers
    int c = 0;
    while (serverlist[c].id != 0) {
      c++;
    }
    //sort that thing :)
    sortprio fsrvs[CONTEXT_MAX_FOUND_SERVERS];
    //find found servers and put them into fsrvs
    for (int i = 0; i < CONTEXT_MAX_FOUND_SERVERS; i++) {
      fsrvs[i].id = 0;
      if (serverlist[i].id != 0) {
        fsrvs[i].id = serverlist[i].id;
        for (int j = 0; j < CONTEXT_MAX_CTYPES; j++) {
          fsrvs[i].prio[j] = 0;
          if (serverlist[i].ct[j] == 1) {
            if (j%2 == 0)
              fsrvs[i].prio[j] = ((data.ct[j/2] >> 4) & 0x7);
            else
              fsrvs[i].prio[j] = ((data.ct[j/2] & 0xF) & 0x7);
          }
        }//sort prios of every server
        for(int k = 0; k < CONTEXT_MAX_CTYPES; k++) {
          for(int l = 0; l < (CONTEXT_MAX_CTYPES - 1); l++) {
            if (fsrvs[i].prio[l] < fsrvs[i].prio[l+1]) {
              uint8_t temp;
              temp = fsrvs[i].prio[l];
              fsrvs[i].prio[l] = fsrvs[i].prio[l+1];
              fsrvs[i].prio[l+1] = temp;
            }
          }
        }
      }
    }
    //sort servers
    for (int l = 0; l < CONTEXT_MAX_CTYPES; l++) {
      for(int m = 0; m < c; m++) {
        for(int n = 0; n < (c - 1); n++) {
          if (fsrvs[n].prio[l] < fsrvs[n+1].prio[l]) {
            ServerList temp;
            temp = serverlist[n];
            serverlist[n] = serverlist[n+1];
            serverlist[n+1] = temp;
          }
        }
      }
    }
    //first server in list = result
    result.id = serverlist[0].id;
    result.sid = 0;
    result.st = data.st;
    if (c > 1)
      result.na = 0;
    else
      result.na = 1;
    memcpy(result.ct, serverlist[0].ct, sizeof(serverlist[0].ct));
    memcpy(result.sl, serverlist, sizeof(serverlist));
    click_chatter("CServers: found best server with %u at id %s na-flag is %d", result.st, IPAddress(result.id).s().c_str(), \
    result.na);
  }
  else {
    result.id = 0;
    result.sid = 0;
    result.st = data.st;
    result.na = 1;
    click_chatter("na-flag is now %d", result.na);
    for (int i=0; i<CONTEXT_MAX_CTYPES; i++) {
      result.ct[i] = 0;
    }
    for (int i=0; i<CONTEXT_MAX_FOUND_SERVERS; i++) {
      result.sl[i].id = 0;
    }
    click_chatter("CServers: no entry was found in the routingtable");
  }

  return result;
}

CRREQResult
CServers::findContextEntry(CRREQFind input){
  CRREQResult result;
  switch(_fmethod) {
    case  0: result = findContextEntryMethod0(input); break;
    case  1: result = findContextEntryMethod1(input); break;
    case  2: result = findContextEntryMethod2(input); break;
    case  3: result = findContextEntryMethod3(input); break;
    default: result = findContextEntryMethod1(input); break;
  }
  return result;
```

```
}

void
CServers::readContextEntry(uint32_t id){
  //search for entry with data.id and put it in read
  ContextMap::Pair* read = contexts.find_pair(id);

  if (read) {
    click_chatter("read ip %u",read->value.id);
    int i = 0;
    int j;

    while (read->value.ca[i].st != 0) {
      click_chatter("read st %u",read->value.ca[i].st);
      for (j=0; j<CONTEXT_MAX_SERVICES; j++) {
        if (read->value.ca[i].ct[j] == true) {
         click_chatter("read ct %u", j+1);
        }
      }
      i++;
    }
    click_chatter("========");
  }
  else {
    click_chatter("no entry with id %u",id);
  }
}

void
CServers::readContextEntryByST(uint16_t st){
  //search for entries with servicetype = st
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    int i = 0;
    while (iter.value().ca[i].st != 0) {
      if (iter.value().ca[i].st == st) {
        click_chatter("CServers: found st %u at id %s",st,IPAddress(iter.value().id).s().c_str());
      }
      i++;
    }
  }
}

void
CServers::deleteoldContextEntries(){
  //delete old entries
  for(ContextMap::const_iterator iter = contexts.begin(); iter != contexts.end(); ++iter){
    //if entry too old, delete it
    Timestamp now;
    now = Timestamp::now();
    uint32_t ts = now.sec()-CONTEXT_SERVERFILES_MAX_AGE;
    if (iter.value().ts < ts) {
      click_chatter("CServers: entry at id %s deleted, because %u < %u", IPAddress(iter.value().id).s().c_str(), iter.value().ts, \
      now.sec()-CONTEXT_SERVERFILES_MAX_AGE);
      contexts.remove(iter.value().id);
      deleteContextFile(iter.value().id);
    }
  }
}

void
CServers::insertContextEntry(struct ctable data){
  //if entry with data.id exists, don't insert again
  //click_chatter("id %u",data.id);
  if (!contexts.find_pair(data.id)) {
    //insert entry with key data.id
    contexts.insert(data.id,data);
    click_chatter("CServers: entry with id %s inserted", IPAddress(data.id).s().c_str());
    //click_chatter("INSERTED1 ip %d with %d types", data.id, sizeof(data.ca));
  }
}

void
CServers::deleteContextFile(uint32_t id) {
  DIR *dir;
  struct dirent *entry;
  char path[255], filename[64];
  uint32_t fid;
  dir = opendir(CONTEXT_SERVERFILES_PATH);
  do {
    entry = readdir(dir);
    if (entry && strstr(entry->d_name,CONTEXT_SERVERFILES_EXTENSION) && ((strstr(entry->d_name,\
    CONTEXT_SERVERFILES_EXTENSION)-entry->d_name+1) == (strlen(entry->d_name)-4))) {
      strcpy(path,CONTEXT_SERVERFILES_PATH);
      strcat(path,entry->d_name);
      memset(filename, '\0', sizeof(filename));
      strncpy(filename, entry->d_name, strlen(entry->d_name)-5);
      fid = strtoul(filename, NULL, 0);
      if (fid == id) {
        remove(path);
        click_chatter("cservers: file %s deleted",path);
      }
    }
  } while(entry);
}

void
CServers::deleteContextEntry(uint32_t id) {
  if (contexts.find_pair(id)) {
    contexts.remove(id);
    click_chatter("CServers: entry with %s deleted", IPAddress(id).s().c_str());
```

```
  }
  deleteContextFile(id);
}

void
CServers::updateContextEntry(struct ctable data){
  ContextMap::Pair* read = contexts.find_pair(data.id);
  contexts.remove(read->value.id);
  contexts.insert(data.id,data);
  click_chatter("CServers: entry with id %s updated", IPAddress(data.id).s().c_str());
}

void
CServers::addContextEntry(){

  //file and dir handlers, some ints and the pathname
  DIR *dir;
  FILE *file;
  int k,l,rowpos,colpos;
  char path[255], filename[64];
  //format for context table entry
  struct ctable ctxt;
  //format for stat.h -> determine date of change
  struct stat buff;
  //format for directory processing
  struct dirent *entry;
  //for current time
  Timestamp now;
  dir = opendir(CONTEXT_SERVERFILES_PATH);

  do {
    entry = readdir(dir);
    if (entry) {

      //if (".list" in filename) && (".list" is at end of filename) do...
      if (strstr(entry->d_name,CONTEXT_SERVERFILES_EXTENSION) && ((strstr(entry->d_name,\
      CONTEXT_SERVERFILES_EXTENSION)-entry->d_name+1) == (strlen(entry->d_name)-4))) {
        //copy "path" to var path and append filename
        strcpy(path,CONTEXT_SERVERFILES_PATH);
        strcat(path,entry->d_name);
        //cut fileextension from filename
        memset(filename, '\0', sizeof(filename));
        strncpy(filename, entry->d_name, strlen(entry->d_name)-5);
        //convert string id to uint32_t
        ctxt.id = strtoul(filename, NULL, 0);

        //determine time of change of the file
        if (stat(path, &buff) == 0) {
          ctxt.ts = buff.st_ctime;
        }

        //if file is older than CONTEXT_SERVERFILES_MAX_AGE, don't put data into routingtable
        //and if id in routingtable, only update lifetime
        now = Timestamp::now();
        uint32_t ts = now.sec()-CONTEXT_SERVERFILES_MAX_AGE;
        if (ctxt.ts > ts) {

          //open file in readonly mode
          file = fopen(path,"r");

          //reset array
          for (k=0; k<CONTEXT_MAX_SERVICES; k++) {
            ctxt.ca[k].st = 0;
            for (l=0; l<CONTEXT_MAX_CTYPES; l++) {
              ctxt.ca[k].ct[l] = false;
            }
          }

          rowpos = 0;
          colpos = 0;
          bool shutdown = false;
          unsigned int buffer;
          char shutdownstr[strlen(CONTEXT_SERVERFILES_SHUTDOWN)+1];

          fgets(shutdownstr, strlen(CONTEXT_SERVERFILES_SHUTDOWN)+1, file);
          if (strcmp(shutdownstr,CONTEXT_SERVERFILES_SHUTDOWN) == 0) {
            click_chatter("CServers: server %s sent shutdown signal", IPAddress(ctxt.id).s().c_str());
            shutdown = true;
          }
          else {
            clearerr(file);
            rewind(file);
            while (fscanf(file, "%u", &buffer) > 0) {
              //if we are on beginning of line (colpos), read the service type @ rowpos
              if (colpos == 0) {
                //click_chatter("server: %s, service: %u, row: %u", IPAddress(ctxt.id).s().c_str(), buffer, rowpos);
                ctxt.ca[rowpos].st = buffer;
              }
              else {
                ctxt.ca[rowpos].ct[buffer-1] = true;
                //click_chatter("server: %s, context: %u, col: %u", IPAddress(ctxt.id).s().c_str(), buffer, colpos);
              }
              colpos++;
              //if end of line do...
              if (fgetc(file) == CONTEXT_SERVERFILES_EOL) {
                rowpos++;
                colpos = 0;
              }
            }
          }
```

```
          fclose (file);

          //if timestamp not expired, update it, else insert
          if (ContextMap::Pair* read = contexts.find_pair(ctxt.id)) {
            if ((read->value.ts < ctxt.ts) && (shutdown == false))
              updateContextEntry(ctxt);
            if (shutdown == true)
              deleteContextEntry(ctxt.id);
          }
          else {
            if (shutdown == false)
              insertContextEntry(ctxt);
            else {
              remove(path);
              click_chatter("CServers: file %s deleted", path);
            }
          }
        }
        else {
          remove(path);
          click_chatter("cservers: file %s deleted", path);
        }

        //reset array for the following entry
        for (k=0; k<CONTEXT_MAX_SERVICES; k++) {
          ctxt.ca[k].st = 0;
          for (l=0; l<CONTEXT_MAX_CTYPES; l++) {
            ctxt.ca[k].ct[l] = false;
          }
        }
      }
    }
  } while (entry);

  closedir(dir);
}

void
CServers::updateContextTable(){
  addContextEntry();
  deleteoldContextEntries();
}

enum { H_FMETHOD };

int
CServers::write_handler(const String &str_in, Element *e, void *thunk, ErrorHandler *errh) {
  String s = cp_uncomment(str_in);
  CServers *cs = static_cast<CServers *>(e);
  if ((uintptr_t)thunk == H_FMETHOD) {
    if (!cp_unsigned(s, (uint32_t *)&cs->_fmethod))
      return errh->error("'fmethod' should be an unsigned integer");
    if (cs->_fmethod > 3) {
      cs->_fmethod = 0;
      return errh->error("'fmethod' should have a value between 0 and 3, setting back to 0", cs->_fmethod);
    }
    click_chatter("CServers: selection method changed to %d", cs->_fmethod);
  }
  return 0;
}

void
CServers::add_handlers()
{
  add_write_handler("fmethod", write_handler, (void *)H_FMETHOD);
}

//macro magic
#include <click/bighashmap.cc>
#if EXPLICIT_TEMPLATE_INSTANCES
template class HashMap<IPPair, void *>;
template class HashMap<IPAddress, int>;
#endif

CLICK_ENDDECLS

EXPORT_ELEMENT(CServers CServers-Context_CServers)
```

ForwardCIP

context_forward_cip.hh

```
#ifndef CONTEXT_FORWARDCIP_HH
#define CONTEXT_FORWARDCIP_HH
#include <click/element.hh>

#include "context_cservers.hh"
#include "context_cclients.hh"

CLICK_DECLS

class ForwardCIP : public Element {
public:

ForwardCIP();
```

```
~ForwardCIP();

const char *class_name() const { return "ForwardCIP"; }
const char *port_count() const  { return "2/2"; }
const char *processing() const { return "h"; }

int configure(Vector<String> &, ErrorHandler *);
        Packet* forward(Packet *, context_ip_fwaddr);
        void push(int, Packet *);

    private:
        CServers* context_table;
        CClients* client_table;

};

CLICK_ENDDECLS
#endif
```

context_forward_cip.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include <clicknet/icmp.h>
#include <clicknet/ip.h>
#include <clicknet/ether.h>
#include <click/packet_anno.hh>
#include <click/packet.hh>

#include "context_forward_cip.hh"

CLICK_DECLS
ForwardCIP::ForwardCIP()
{
}

ForwardCIP::~ForwardCIP()
{
}

int
ForwardCIP::configure(Vector<String> &conf, ErrorHandler *errh)
{
  Element* context_table_element;
  Element* client_table_element;
  if (cp_va_parse(conf, this, errh,
      cpElement, "CServers", &context_table_element,
      cpElement, "CClients", &client_table_element,
      cpEnd) < 0) {
    return -1;
  }
  if (strcmp(context_table_element->class_name(), "CServers") != 0){
    errh->error("Supplied Element is not CServers");
    return -1;
  }
  if (!(context_table=(CServers*)context_table_element)){
    errh->error("Supplied element is not a valid CServers element (cast failed)");
    return -1;
  }
  if (strcmp(client_table_element->class_name(), "CClients") != 0){
    errh->error("Supplied Element is not CClients");
    return -1;
  }
  if (!(client_table=(CClients*)client_table_element)){
    errh->error("Supplied element is not a valid CClients element (cast failed)");
    return -1;
  }
  return 0;
}

Packet*
ForwardCIP::forward(Packet *p_in, context_ip_fwaddr addr_in){
  assert(p_in);
  //context_ip_header *cip_in = (struct context_ip_header *) p_in->network_header();
  WritablePacket *p_out = p_in->uniqueify();
  context_ip_header* cip_out = (struct context_ip_header*) p_out->data();

  cip_out->dst = addr_in.dst;
  cip_out->src = addr_in.src;
  cip_out->oid = addr_in.id;
  cip_out->chksum = 0;
  cip_out->chksum += click_in_cksum((unsigned char *)cip_out, sizeof(struct context_ip_header));
  //click_chatter("packet from %x to %x with id %x\npacket forw %x to %x with id %x", cip_in->src, cip_in->dst, cip_in->oid, \
  cip_out->src, cip_out->dst, cip_out->oid);
  p_out->set_dst_ip_anno(IPAddress(cip_out->dst));
  p_out->set_network_header(reinterpret_cast<const unsigned char *>(cip_out), sizeof(struct context_ip_header));

  return p_out;
  p_in->kill();
}

void
ForwardCIP::push(int port, Packet *p_in){
  assert(port >= 0 && port <= 1);
  assert(p_in);
```

```
  Packet *p_out;
  //incoming packets from input0 (from eth)
  if (port == 0) {
    context_ip_header *cip_in = (struct context_ip_header *) p_in->network_header();
    context_ip_fwaddr addr;
    //search in CClients for entry (incoming packet from client)
    ResultClientCRREQ resultclient = client_table->findClientEntry(IPAddress(cip_in->src).addr(), ntohs(cip_in->osid));
    //search in CClients for entry (incoming packet from server)
    ResultClientCRREQ resultserver = client_table->findClientEntry(cip_in->oid, ntohs(cip_in->osid));
    //if entry exists, forward packet to server from CClients table
    if (resultclient.id != 0 || resultserver.id != 0) {
      if (resultclient.id != 0) {
        click_chatter("CIPForward: entry found, forwarding to server %s", IPAddress(resultclient.ls).s().c_str());
        addr.src = cip_in->dst;
        addr.dst = IPAddress(resultclient.ls);
        addr.id = resultclient.id;
      }
      //if entry exists, forward packet to client with original server from CClients table
      else {
        click_chatter("CIPForward: entry found, forwarding to client %s", IPAddress(resultserver.id).s().c_str());
        addr.src = cip_in->dst;
        addr.dst = IPAddress(resultserver.id);
        addr.id = resultserver.os;
      }
    }
    //if no entry exists, stupidly forward the packet
    else {
      click_chatter("CIPForward: no entry found, stupid forwarding from %s to %s", IPAddress(cip_in->src).s().c_str(), \
      IPAddress(cip_in->oid).s().c_str());
      addr.src = cip_in->dst;
      addr.dst = IPAddress(cip_in->oid);
      addr.id = IPAddress(cip_in->src).addr();
    }
    //push to output0
    p_out = forward(p_in, addr);
    output(0).push(p_out);
  }
  //incoming packets from input1 (last packet from queue, when discovery failed at waitingfordiscovery)
  else {
    context_ip_header *cip_in = (struct context_ip_header *) p_in->network_header();
    //destination unreachable -> delete it from CServers
    context_table->deleteContextEntry(IPAddress(cip_in->dst).addr());
    context_table->updateContextTable();
    //look, if client wants rerouting (entry in CClients exists?)
    ResultClientCRREQ findclient = client_table->findClientEntry(cip_in->oid, ntohs(cip_in->osid));
    if ((findclient.id != 0) && (findclient.nr == 0)) {
      //if client was found in CClients, search for id in CServers
      //click_chatter("CIPForward: found entry in cclients, searching for alternative server");
      uint32_t newserver;
      if (context_table->_fmethod == 0) {
        CRREQFind input;
        input.id = findclient.ls;
        input.st = findclient.st;
        ResultClientOSCT osctresult = client_table->findOSCT(findclient.id, findclient.os, findclient.st);
        memcpy(input.osct, osctresult.osct, sizeof(osctresult.osct));
        memcpy(input.ct, findclient.ct, sizeof(findclient.ct));
        CRREQResult result = context_table->findRerouteContextEntry(input);
        newserver = result.id;
      }
      else {
        FindNxtServer find;
        find.server = findclient.ls;
        find.client = cip_in->oid;
        click_chatter("server %s client %x", IPAddress(findclient.ls).s().c_str(), find.client);
        ServerList srv;
        srv = client_table->findNextServer(find);
        newserver = srv.id;
        click_chatter("newserver %s",IPAddress(newserver).s().c_str());
      }
      //if server found -> update CClients-Entry, modify packet and push it to output0
      if (newserver != 0) {
        click_chatter("CIPForward: found alternative server %s, forwarding packet with src %s dst %s id %s to network", \
        IPAddress(newserver).s().c_str(), IPAddress(cip_in->src).s().c_str(), IPAddress(newserver).s().c_str(), \
        IPAddress(cip_in->oid).s().c_str());
        client_table->updateClientEntry(cip_in->oid, findclient.ls, newserver);
        context_ip_fwaddr addr;
        addr.src = cip_in->src;
        addr.dst = IPAddress(newserver);
        addr.id = cip_in->oid;
        p_out = forward(p_in, addr);
        output(0).push(p_out);
      }
      else {
        click_chatter("CIPForward: did not find any alternative server, generating icmp error");
        output(1).push(p_in);
      }
    }
    else {
      click_chatter("CIPForward: unknown client or no reoruting allowed, generating icmp error");
      output(1).push(p_in);
    }
  }
}

CLICK_ENDDECLS

EXPORT_ELEMENT(ForwardCIP ForwardCIP-Context_ForwardCIP)
```

GenerateCRREP

context_generate_crrep.hh

```
#ifndef CONTEXT_GENERATECRREP_HH
#define CONTEXT_GENERATECRREP_HH
#include <click/element.hh>

#include "context_cservers.hh"
#include "context_cclients.hh"
#include "aodv_neighbours.hh"
#include "aodv_setrrepheaders.hh"

CLICK_DECLS


class GenerateCRREP : public Element { public:

    GenerateCRREP();
    ~GenerateCRREP();

    const char *class_name() const      { return "GenerateCRREP"; }
    const char *port_count() const      { return "1/1"; }
    const char *processing() const      { return "h"; }

    int configure(Vector<String> &, ErrorHandler *);

    Packet* generate(CRREQResult, Packet *);
    void push(int, Packet *);

  private:

    CServers* context_table;
    CClients* client_table;
    AODVNeighbours* neighbour_table;
    AODVSetRREPHeaders* setrrepheaders;
    const IPAddress * myIP;
    uint8_t NumIF;
    IPAddress MyIP[AODV_MAX_IF];
};

CLICK_ENDDECLS
#endif
```

context_generate_crrep.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include <click/packet_anno.hh>
#include <clicknet/ip.h>
#include <clicknet/ether.h>
#include <clicknet/udp.h>
#include <arpa/inet.h>

#include "context_generate_crrep.hh"
#include "aodv_broadcastheader.hh"

CLICK_DECLS
GenerateCRREP::GenerateCRREP():
    context_table(0),
    neighbour_table(0)
{
}

GenerateCRREP::~GenerateCRREP()
{
}

int
GenerateCRREP::configure(Vector<String> &conf, ErrorHandler *errh)
{
    Element* context_table_element;
    Element* client_table_element;
    Element* neighbour_table_element;
    Element* setrrepheaders_element;
    //copy elements
    if (cp_va_parse(conf, this, errh,
        cpElement, "CServers", &context_table_element,
        cpElement, "CClients", &client_table_element,
        cpElement, "AODVNeighbours", &neighbour_table_element,
        cpElement, "AODVSetRREPHeaders", &setrrepheaders_element,
        cpEnd) < 0) {
      return -1;
    }
    if (strcmp(context_table_element->class_name(), "CServers") != 0){
      errh->error("Supplied Element is not CServers");
      return -1;
    }
    if (!(context_table=(CServers*)context_table_element)){
      errh->error("Supplied element is not a valid CServers element (cast failed)");
      return -1;
    }
    if (strcmp(client_table_element->class_name(), "CClients") != 0){
```

```
      errh->error("Supplied Element is not CClients");
      return -1;
    }
    if (!(client_table=(CClients*)client_table_element)){
      errh->error("Supplied element is not a valid CClients element (cast failed)");
      return -1;
    }
    if (strcmp(neighbour_table_element->class_name(), "AODVNeighbours") != 0){
      errh->error("Supplied Element is not AODVNeighbours");
      return -1;
    }
    if (!(neighbour_table=(AODVNeighbours*)neighbour_table_element)){
      errh->error("Supplied element is not a valid AODVNeighbours element (cast failed)");
      return -1;
    }
    if (strcmp(setrrepheaders_element->class_name(), "AODVSetRREPHeaders") != 0){
        errh->error("Supplied element is not an AODVSetRREPHeaders element but a %s",setrrepheaders_element->class_name());
        return -1;
    }
    if (!(setrrepheaders=(AODVSetRREPHeaders*)setrrepheaders_element)){
        errh->error("Supplied element is not a valid AODVSetRREPHeaders element (cast failed)");
        return -1;
    }
    NumIF = neighbour_table->getNumIF();
    for (uint8_t i = 0; i < NumIF; i++) {
      MyIP[i] = neighbour_table->getIPbyID(i);
    }
    return 0;
}

Packet*
GenerateCRREP::generate(CRREQResult data, Packet * p_in){
  int tailroom = 0;
  int headroom = sizeof(click_ether) + sizeof(click_ip) + sizeof(click_udp);
  int crrepsize = sizeof(struct aodv_crrep_header);

  aodv_crreq_header *crreq_in = (aodv_crreq_header*) (p_in->data() + headroom);

  WritablePacket *p_out = Packet::make(headroom, 0, crrepsize, tailroom);
  if (!p_out) return 0;

  memset(p_out->data(), 0, p_out->length());
  aodv_crrep_header* crrep_out = (struct aodv_crrep_header*) p_out->data();

  //am i the destination?
  bool imdestination = false;
  for (uint8_t i = 0; i < NumIF; i++) {
    if (crreq_in->destination == MyIP[i])
      imdestination = true;
  }
  if (imdestination) {
    neighbour_table->updateMySequenceNumber(ntohl(crreq_in->destinationseqnr)); //RFC 6.1
  }

  //regular rrep header
  crrep_out->type = CONTEXT_AODV_RREP;
  crrep_out->ranreserved = 0;
  crrep_out->ranreserved += CONTEXT_RREP_R << 7;  //r-flag
  crrep_out->ranreserved += CONTEXT_RREP_A << 6;  //a-flag
  crrep_out->ranreserved += data.na << 5;         //no-alternatives-flag
  crrep_out->reservedprefixsz = 0;
  crrep_out->reservedprefixsz += CONTEXT_RREP_RESERVED << 4;    //reserved field
  crrep_out->reservedprefixsz += CONTEXT_RREP_PRFXSIZE & 0xFF;  //prefix size
  crrep_out->destination = crreq_in->destination;
  if (imdestination){
    crrep_out->destinationseqnr = htonl(neighbour_table->getMySequenceNumber());
    crrep_out->lifetime = htonl(CONTEXT_SERVERFILES_MAX_AGE);
    crrep_out->hopcount = 0;
  } else {
    uint32_t * destinationseqnr = neighbour_table->getSequenceNumber(crreq_in->destination);
    assert(destinationseqnr); // if we don't have a sequence number why are we responding...
    crrep_out->destinationseqnr = htonl(*destinationseqnr);
    crrep_out->lifetime = htonl(neighbour_table->getLifetime(crreq_in->destination));
    crrep_out->hopcount = neighbour_table->getHopcount(crreq_in->destination);
    delete destinationseqnr;
  }
  crrep_out->originator = crreq_in->originator;

  //rrep header extension
  crrep_out->xtype = CONTEXT_RREP_XTYPE;
  crrep_out->length = sizeof(crrep_out->ident) + sizeof(crrep_out->service) + sizeof(crrep_out->contexttype) + \
  sizeof(crrep_out->sessid);
  crrep_out->sessid = htons(data.sid);
  crrep_out->ident = data.id;
  crrep_out->service = htons(data.st);
  int i,j,k;
  for (i = 0; i < CONTEXT_MAX_CTYPES; i++) {
    j = i / 8;
    k = i % 8;
    if (data.ct[i] == true) {
      crrep_out->contexttype[j] += 1 << (7-k);
    }
    else {
      crrep_out->contexttype[j] += 0 << (7-k);
    }
  }

  p_in->kill();
  return p_out;
}
```

```
void
GenerateCRREP::push (int port, Packet * p_in){
  assert(port == 0);
  Packet* p_out;

  int headroom = sizeof(click_ether) + sizeof(click_ip) + sizeof(click_udp);
  aodv_crreq_header *crreq_in = (aodv_crreq_header*) (p_in->data() + headroom);
  const click_ip * ip_header = p_in->ip_header();

  if (ntohs(crreq_in->service) != 0) {
    //update routing tables
    uint8_t ifid;
    ifid = PAINT_ANNO(p_in);
    IPAddress myip;
    myip = IPAddress(neighbour_table->getIPbyID(ifid));

    neighbour_table->updateRoutetableEntry(ip_header->ip_src,1, ip_header->ip_src, ifid);
    context_table->updateContextTable();
    //context_table->addContextEntry();
    //context_table->deleteoldContextEntries();

    //deliver data for table query
    CRREQFind input;
    input.st = ntohs(crreq_in->service);
    input.id = (crreq_in->ident[1] << 16) + crreq_in->ident[0];
    memcpy(input.ct, crreq_in->context, sizeof(input.ct));

    CRREQResult result;
    if ((context_table->_fmethod > 0) && (input.id != 0)) {
      FindNxtServer find;
      find.client = IPAddress(crreq_in->originator);
      find.server = input.id;
      ServerList srv;
      srv = client_table->findNextServer(find);
      result.id = srv.id;
      result.st = input.st;
      result.na = srv.na;
      memcpy(result.ct, srv.ct, sizeof(srv.ct));
    }
    else
      result = context_table->findContextEntry(input);

    //insert crreq query in routingtable and determine session id
    //if server was found
    if ((result.id != 0)) {
      FindClientCRREQ clientdata;
      clientdata.id = IPAddress(crreq_in->originator).addr();
      clientdata.nr = (crreq_in->jrgdureserved >> 2) & 0x1;
      clientdata.st = input.st;
      clientdata.sid = client_table->determineSessionID(clientdata.id);
      result.sid = clientdata.sid;
      //click_chatter("genrrep: genrrep sid: %u",clientdata.sid);
      Timestamp now = Timestamp::now();
      clientdata.ts = now.sec();
      clientdata.ls = result.id;
      memcpy(clientdata.osct, result.ct, sizeof(result.ct));
      memcpy(clientdata.ct, input.ct, sizeof(input.ct));
      memcpy(clientdata.sl, result.sl, sizeof(result.sl));
      client_table->insertClientEntry(clientdata);
    }

    //generate crrep
    p_out = generate(result, p_in);

    //click_chatter("originator_ip %s %x",IPAddress(crreq_in->originator).s().c_str(),crreq_in->originator);
    IPAddress* nexthop = neighbour_table->nexthop(IPAddress(crreq_in->originator));
    assert(nexthop);
    setrrepheaders->addRREP(p_out,nexthop);

    Packet* buff = AODVBroadcastHeader::setBroadcastHeader(p_out,myip,250);

    output(0).push(buff);
//     p_out->kill();
  }
  else {
    click_chatter("genrrep: wrong packet received");
  }

  //p_in->kill();
}

CLICK_ENDDECLS

EXPORT_ELEMENT(GenerateCRREP GenerateCRREP-Context_GenerateCRREP)
```

Abkürzungsverzeichnis

AAL ..... **A**TM **A**daptation **L**ayer
ABR ..... **A**ssociativity-**B**ased **R**outing
AFH ..... **A**daptive **F**requency **H**opping
AODV ..... **A**d hoc **O**n-Demand **D**istance **V**ector
AODV-SD ..... **AODV-S**ervice **D**iscovery
AODV-UU ..... **AODV**-Softwareimplementierung der **U**niversität **U**ppsala
API ..... **A**pplication **P**rogramming **I**nterface
ARP ..... **A**ddress **R**esolution **P**rotocol
ASCII ..... **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
ATM ..... **A**synchronous **T**ransfer **M**ode
BGP ..... **B**order **G**ateway **P**rotocol
BNEP ..... **B**luetooth **N**etwork **E**ncapsulation **P**rotocol
C ..... **C**opied Flag
CBR ..... **C**omponent-**B**ased **R**outing Protocol
CBRP ..... **C**luster **B**ased **R**outing **P**rotocol
CEDAR ..... **C**ore **E**xtraction **D**istributed **A**d hoc **R**outing
CGSR ..... **C**luster-**H**ead **G**ateway **S**witch **R**outing
CMMBCR ..... **C**onditional **M**ax-**M**in **B**attery **C**apacity **R**outing
CORBA ..... **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
CPU ..... **C**entral **P**rocessing **U**nit
CSMA/CA ..... **C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance
CSMA/CD ..... **C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **D**etection
Ct ..... **C**ontext **t**ype
DAML+OIL ..... **D**arpa **A**gent **M**arkup **L**anguage + **O**ntology **I**nference **L**ayer
DFG ..... **D**eutsche **F**orschungs**G**emeinschaft
DHCP ..... **D**ynamic **H**ost **C**ontrol **P**rotocol
DNS ..... **D**omain **N**ame **S**ystem
DNS-SD ..... **DNS**-Based **S**ervice **D**iscovery
DSDV ..... **D**estination-**S**equenced **D**istance-**V**ector
DSL ..... **D**igital **S**ubscriber **L**ine
DSR ..... **D**ynamic **S**ource **R**outing
EDGE ..... **E**nhanced **D**ata Rates for **G**SM **E**volution
EGNOS ..... **E**uropean **G**eostationary **N**avigation **O**verlay **S**ystem
ETSI ..... **E**uropean **T**elecommunications **S**tandards **I**nstitute
F ..... **F**lag
FSR ..... **F**isheye **S**tate **R**outing
GIA ..... **G**lobal **I**P-**A**nycast
GloMoSim ..... **Glo**bal **Mo**bile Information Systems **Sim**ulation Library
GPRS ..... **G**eneral **P**acket **R**adio **S**ervice
GPS ..... **G**lobal **P**ositioning **S**ystem
GSD ..... **G**roup-based **S**ervice **D**iscovery Protocol
GSM ..... **G**lobal **S**ystem for **M**obile Communications
GUI ..... **G**raphical **U**ser **I**nterface

HIPERLAN ..... **HI**gh **PER**formance **L**ocal **A**rea **N**etwork
HSCSD ..... **H**igh **S**peed **C**ircuit **S**witched **D**ata
HTML ..... **H**yper**T**ext **M**arkup **L**anguage
IANA ..... **I**nternet **A**ssigned **N**umbers **A**uthority
ICMP ..... **I**nternet **C**ontrol **M**essage **P**rotocol
Id ..... **Id**entifikator bzw. **Id**entifier
IEEE ..... **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF ..... **I**nternet **E**ngineering **T**ask **F**orce
IMEP ..... **I**nternet **M**ANET **E**ncapsulation **P**rotocol
IN ..... **I**ntelligentes **N**etz
IOS ..... **I**nternetwork **O**perating **S**ystem
IP ..... **I**nternet **P**rotocol
IPv4 ..... **I**nternet **P**rotocol **v**ersion **4**
IPv6 ..... **I**nternet **P**rotocol **v**ersion **6**
ISDN ..... **I**ntegrated **S**ervices **D**igital **N**etwork
ISM ..... **I**ndustrial, **S**cientific, and **M**edical
ISO ..... International Organization for Standardization (von gr. „isos“)
JunOS ..... **Jun**iper **O**perating **S**ystem
JVM ..... **J**ava **V**irtual **M**achine
k. A. ..... **k**eine **A**ngabe
kt ..... ein einzelner **Kon**extytp
KtC ..... **K**ontext**t**ypen durch den **C**lient bereitgestellt
KtS ..... **K**ontext**t**ypen vom **S**erver unterstützt
L2CAP ..... **L**ogical **L**ink **C**ontrol and **A**daption **P**rotocol
LBS ..... **L**ocation **B**ased **S**ervice
LLC ..... **L**ogical **L**ink **C**ontrol
LMR ..... **L**ightweight **M**obile **R**outing
LSD ..... **L**ightweight **S**ervice **D**iscovery
LUS ..... **L**ook**U**p **S**ervice
MAC ..... **M**edium **A**ccess **C**ontrol
MANET ..... **M**obile **A**d Hoc **NET**work
mDNS ..... **m**ulticast **DNS**
MSC ..... **M**essage **S**equence **C**hart
NA ..... **N**o **A**lternatives
nam ..... **n**etwork **ani**m**a**tor
NR ..... **N**o **R**eroute
ns-2 ..... **n**etwork **s**imulator version **2**.xx
OC ..... **O**ption **C**lass
OLSR ..... **O**timized **L**ink **S**tate **R**outing
OMNeT++ ..... **O**bjective **M**odular **Ne**twork **T**estbed in C++
ON ..... **O**ption **N**umber
OSGi ..... **O**pen **S**ervices **G**ateway **i**nitiative
OSI ..... **O**pen **S**ystems **I**nterconnection
OSPF ..... **O**pen **S**hortest **P**ath **F**irst
OTcl ..... **O**bject **T**ool **c**ommand **l**anguage
P ..... **P**riorität bzw. **P**riority
PAN ..... **P**ersonal **A**rea **N**etwork
PARO ..... **P**ower-**A**ware **R**outing **O**ptimization
PC ..... **P**ersonal **C**omputer
PCF ..... **P**oint **C**oordination **F**unction
PCI ..... **P**eripheral **C**omponent **I**nterconnect
PDA ..... **P**ersonal **D**igital **A**ssistent
POTS ..... **P**lain **O**ld **T**elephone **S**ervice
PPP ..... **P**oint-to-**P**oint **P**rotocol

PPPoE ..... **PPP o**ver **E**thernet
QoS ..... **Q**uality **o**f **S**ervice
R ..... **R**eichweite
RAM ..... **R**andom **A**ccess **M**emory
RBR ..... **R**egion-**B**ased **R**outing
RDMAR ..... **R**elative **D**istance **M**icro-discovery **A**d hoc **R**outing
RERR ..... **R**oute **ERR**or
RFID ..... **R**adio **F**requency **ID**entification
RIP ..... **R**outing **I**nformation **P**rotocol
RLC ..... **R**adio **L**ink **C**ontrol
RMI ..... **R**emote **M**ethod **I**nvocation
RREP ..... **R**oute **REP**ly
RREP ACK ..... **R**oute **REP**ly **ACK**nowledgement
RREQ ..... **R**oute **REQ**uest
RTS/CTS ..... **R**equest **T**o **S**end/**C**lear **T**o **S**end
SDP ..... **S**ervice **D**iscovery **P**rotocol
SIG ..... **S**pecial **I**nterest **G**roup
SLP ..... **S**ervice **L**ocation **P**rotocol
SM ..... **S**alutation **M**anager
SNDCP ..... **S**ubNetwork **D**ependent **C**onvergence **P**rotocol
SONET ..... **S**ynchronous **O**ptical **NET**work
SSDP ..... **S**imple **S**ervice **D**iscovery **P**rotocol
SSH ..... **S**ecure **SH**ell
SSR ..... **S**ignal **S**tability **R**outing
STAR ..... **S**ource **T**ree **A**daptive **R**outing
TAS ..... **T**ouristisches **A**ssistenz **S**ystem
TBRPF ..... **T**opology Dissemination **B**ased on **R**everse- **P**ath
TCP ..... **T**ransmission **C**ontrol **P**rotocol
TORA ..... **T**eporally-**O**rdered **R**outing **A**lgorithm
UDP ..... **U**ser **D**atagram **P**rotocol
UMTS ..... **U**niversal **M**obile **T**elecommunications **S**ervice
UPnP ..... **U**niversal **P**lug a**n**d **P**lay
URL ..... **U**niform **R**esource **L**ocator
UWB ..... **U**ltra **W**ide **B**and
WiMAX ..... **W**orldwide **i**nteroperability for **M**icrowave **ACC**ess
WLAN ..... **W**ireless **L**ocal **A**rea **N**etwork
WNTE ..... **W**ireless **N**etwork **T**opology **E**mulator
WRP ..... **W**ireless **R**outing **P**rotocol
X ..... prozentualer Vergleichswert
XML ..... E**X**tensible **M**arkup **L**anguage
ZRP ..... **Z**one **R**outing **P**rotocol

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[AcSB99] J. J. Garcia-Luna Aceves, M. Spohn und D. Beyer. Source Tree Adaptive Routing (STAR) Protocol. Internet Draft, IETF MANET Working Group, Oktober 1999.

[AdhR07] List of ad-hoc routing protocols. http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list, März 2007.

[AgTa99a] G. Aggelou und R. Tafazolli. RDMAR: A Bandwidth-Efficient Routing Protocol for Mobile Ad Hoc Networks. In *Proceedings of the ACM International Workshop on Wireless Mobile Multimedia WOWMOM*, Seattle, USA, August 1999. S. 26–33.

[AgTa99b] G. Aggelou und R. Tafazolli. Relative Distance Micro-discovery Ad Hoc Routing (RDMAR) Protocol. Internet Draft, Mobile Ad Hoc Networking Working Group, September 1999.

[ALSS03] S. Abeck, P. C. Lockemann, J. Schiller und J. Seitz. *Verteilte Informationssysteme*. dpunkt.verlag GmbH, Heidelberg. ISBN 3-89864-188-0, 2003.

[BaAg07] H. Badis und K. A. Agha. Quality of Service for Ad hoc Optimized Link State Routing Protocol (QOLSR). Internet Draft, IETF MANET Working Group, März 2007.

[Blue04] Specification of the Bluetooth System – Bluetooth Specification Version 2.0 + EDR. http://german.bluetooth.com/Bluetooth/Learn/Technology/Specifications/, Bluetooth SIG, Inc., November 2004.

[Blue07] Bluetooth Wireless-Technologie – Profile. http://german.bluetooth.com/Bluetooth/Learn/Works/Profiles_Overview.htm, Bluetooth SIG, Inc., Mai 2007.

[Born01] C. Bornträger. Ad-hoc-Netzwerke. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juli 2001.

[Born02] C. Bornträger. Simulation der Wegewahl in Ad-hoc Netzen. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juni 2002.

[Born03] C. Bornträger. Contextual Influence on the Usability of Different Media Types. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Februar 2003.

[Brae05] B. Braem. Implementatie en Evaluatie van Ad-Hoc On-Demand Distance Vector Routing. Masterarbeit, Universiteit Antwerpen, Faculteit Wetenschappen, Departement Wiskunde-Informatica, Antwerpen, Belgien, Juni 2005.

[BrBC97] P. Brown, J. Bovey und X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications* 4(5), 1997, S. 58–64.

[Böri02] R. Böringer. Analyse und Konzeption einer Architektur zur kontextsensitiven Diensterbringung. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2002.

[BRPD03] E. M. Belding-Royer, C. E. Perkins und S. R. Das. RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing. Request for Comments 3561, Network Working Group, Juli 2003. Category: Experimental.

[CCGe97] W. Liu C. Chiang, H. Wu und M. Gerla. Routing in Clustered Multihop, Mobile Wireless Networks. In *Proceedings IEEE SICON'97*, April 1997, S. 197–211.

[ChGZ98] C. Chiang, M. Gerla und L. Zhang. Adaptive Shared Tree Multicast in Mobile Wireless Networks. In *Proceedings of GLOBECOM '98*, November 1998, S. 1817–1822.

[ChKr06a] S. Cheshire und M. Krochmal. DNS-Based Service Discovery. Internet Draft, Apple Computer, Inc., August 2006. Category: Standards Track.

[ChKr06b] S. Cheshire und M. Krochmal. Multicast DNS. Internet Draft, Apple Computer, Inc., Februar 2006. Category: Standards Track.

[ChMi01] S. Chakrabarti und A. Mishra. QoS Issues in Ad Hoc Wireless Networks. *IEEE Communications Magazine* 39(2), Februar 2001, S. 142–148.

[ChTG97] T. Chen, J. Tsai und M. Gerla. QoS Routing Performance in Multihop, Multimedia, Wireless Networks. In *Proceedings of IEEE ICUPC*, Band 2, 1997, S. 557–561.

[CJFY02] D. Chakraborty, A. Joshi, T. Finin und Y. Yesha. GSD: A novel groupbased service discovery protocol for MANETS. In *4th International Workshop on Mobile and Wireless Communications Networks (MWCN 2002)*, September 2002, S. 140–144.

[CMRP07] The Click Modular Router Project. http://pdos.csail.mit.edu/click/, Februar 2007.

[CoDG06] A. Conta, S. Deering und M. Gupta. RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. Request for Comments 4443, Network Working Group, März 2006. Category: Standards Track.

[CoEp95] M. S. Corson und A. Ephremides. A distributed routing algorithm for mobile wireless networks. In *Wireless Networks, ISSN 1022-0038*, Band 1(1), S. 61–81. Springer Netherlands, März 1995.

[CoGi07] M. Conti und S. Giordano. Multihop Ad Hoc Networking: The Theory. *IEEE Communications Magazine* 45(4), April 2007, S. 78–86.

[Cons00] Salutation Consortium. Salutation Architecture Specification Version 2.1. Salutation specification, Salutation Consortium, Utah, USA, April 2000.

[cygw08] Cygwin Information and Installation. http://www.cygwin.com/, Januar 2008.

[DeAb97] A. Dey und G. Abowd. CyberDesk, The Use of Perception in Context-Aware Computing. In *Proceedings of 1997 Workshop on Perceptual User Interfaces PUI '97*, Oktober 1997, S. 26–27.

[DeAb99] A. Dey und G. Abowd. Toward a Better Understanding of Context and Context Awareness. Technischer Bericht Georgia GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, Atlanta, 1999.

[Debe07] M. Debes. Kontextsensitives Routing: Architekturkonzept. Fachpublikation – Forschungsbericht, Digitale Bibliothek Thüringen, http://www.db-thueringen.de/servlets/DocumentServlet?id=8273, Juni 2007.

[Debe08] M. Debes. Modifizierte AODV-Click-Elemente des Kontextrouters – Interner technischer Arbeitsbericht. Technischer Bericht, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, März 2008.

[Deer91] S. Deering. RFC 1256: ICMP Router Discovery Messages. Request for Comments 1256, Network Working Group, September 1991. Category: Standards Track.

[DefD07] Dienstleistung. http://de.wikipedia.org/wiki/Dienstleistung, Oktober 2007.

[DeHi98] S. Deering und R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) Specification. Request for Comments 2460, Network Working Group, Dezember 1998. Category: Standards Track.

[DeLS05] M. Debes, A. Lewandowska und J. Seitz. Definition and Implementation of Context Information. In K. Kyamakya, K. Jobmann und H.-P. Kuchenbecker (Hrsg.), *Joint second Workshop on Positioning, Navigation and Communication 2005 (WPNC '05) & first Ultra-Wideband Expert Talk 2005 (UET '05)*, Band 0.2 der *Hannoversche Beiträge zur Nachrichtentechnik*, Aachen, März 2005. IANT, University of Hannover, Shaker Verlag, S. 63–68. ISBN 3-8322-3746-1.

[DeSe03] M. Debes und J. Seitz. Context-Sensitive Routing in Mobile and Ad-Hoc Networks. In *48. Internationales Wissenschaftliches Kolloquium IWK 2003*, Ilmenau, Germany, 22.–25. September 2003. Technische Universität Ilmenau, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Ilmenau (Thüringen).

[DeSW04] M. Debes, J. Seitz und S. Winter. Positioning in an IEEE 802.11 Wireless Local Area Network. In *49. Internationales Wissenschaftliches Kolloquium IWK 2004*, Ilmenau, Germany, 27.–30. September 2004. Technische Universität Ilmenau, Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau (Thüringen).

[Dey00a] A. Dey. Enabling the Use in Interactive Apllications. In *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness*, The Hague, The Netherlands, April 2000. Doctoral Consortium paper in the Proceedings of the 2000 Conference on Human Factors in Computer Systems (CHI 2000), S. 79–80.

[Dey00b] A. Dey. *Providing Architectural Support for Building Context-Aware Applications*. Dissertation, College of Computing, Georgia Institute of Technology, Dezember 2000.

[Dey01] A. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing Journal* 5(1), 2001, S. 4–7.

[DHST03] M. Debes, M. Heubach, J. Seitz und R. Tosse. Information Without Barriers – A New Form of Ubiquitous Computing. In *48. Internationales Wissenschaftliches Kolloquium IWK 2003*, Ilmenau, Germany, 22.–25. September 2003. Technische Universität Ilmenau, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Ilmenau (Thüringen).

[Diet05] W. Dietz. Location Based Services (LBS) in der Mobilkommunikation. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Oktober 2005.

[Domb07] L. Dombissi. Server for Context Sensitive Routing. Praktikumsbericht, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juni 2007.

[DRWT97] R. Dube, C. D. Rais, K.-Y. Wang und S. K. Tripathi. Signal Stability based Adaptive Routing for Ad-Hoc Mobile Networks. *IEEE Personal Communications*, Februar 1997, S. 36–45.

[Eber06] J. Eberle. Netzplanung von heterogenen taktischen Netzen. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juli 2006.

[Edwa06] W. K. Edwards. Discovery Systems in Ubiquitous Computing. *IEEE Pervasive Computing – Mobile and Ubiquitous Systems* 5(2), April–Juni 2006, S. 70–77.

[ETSI98] ETSI SMG 2. *Universal Mobile Telecommunication System (UMTS); Selection procedures for the choice of radio transmission technologies of the UMTS*, April 1998. UMTS Technical Report TR 101 112 (UMTS 30.03), Version 3.2.0.

[Ever04] F. Evers. Unterstützung eines vertikalen Handovers zwischen GPRS und WLAN durch einen Proxy. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2004.

[Fenn06] B. Fenner. RFC 4727: Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers. Request for Comments 4727, Network Working Group, November 2006. Category: Standards Track.

[Förs05] K. Förster. Ausarbeitung eines Praktikumsversuchs zum Thema Bluetooth. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juni 2005.

[GCNB01] J. Gomez, A. T. Campbell, M. Naghshineh und C. Bisdikian. PARO: A Power-Aware Routing Optimization Scheme for Mobile Ad hoc Networks. Internet Draft, IETF MANET Working Group, Februar 2001.

[GeDü07] L. Geiger und F. Dürr. Kontextbezogene Kommunikation. In J. Roth, A. Küpper und C. Linnhoff-Popien (Hrsg.), *4. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste*. München: Verlag Dr. Hut, September 2007, S. 22–26. ISBN: 978-3-89963-591-1.

[Gens06] H. Gensicke. Location-Based Services auf mobilen Endgeräten im Automotive-Bereich. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, August 2006.

[GePH02] M. Gerla, G. Pei und X. Hong. Fisheye State Routing Protocol (FSR) for Ad Hoc Networks. Internet Draft, IETF MANET Working Group, Juni 2002.

[GloM08] GloMoSim – Global Mobile Information Systems Simulation Library. http://pcl.cs.ucla.edu/projects/glomosim/obtaining_glomosim.html, Januar 2008.

[GMTo05] J. A. Garcia-Macias und D. A. Torres. Service Discovery in Mobile Ad-hoc Networks: Better at the Network Layer? In *Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05.* IEEE Computer Society, 2005, S. 452–457.

[Gome02] Javier Gomez-Castellanos. *Energy-Efficient Routing and Control Mechanisms for Wireless Ad Hoc Networks.* Dissertation, Columbia University, New York, USA, Dezember 2002.

[GPVD99] E. Guttman, C. Perkins, J. Veizades und M. Day. RFC 2608: Service Location Protocol, Version 2. Request for Comments 2608, Network Working Group, Juni 1999. Category: Standards Track.

[Gren02] L. Grenzendörfer. Untersuchungen zu Bluetooth. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, März 2002.

[Gren04] L. Grenzendörfer. Sicherheit in Ad-hoc-Netzen. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, August 2004.

[Guér87] R. A. Guérin. Channel Occupancy Time Distribution in a Cellular Radio System. *IEEE Transaction on Vehicular Technology* 35(3), August 1987, S. 89–99.

[GuPK99] E. Guttman, C. E. Perkins und J. Kempf. RFC 2609: Service Templates and Service: Schemes. Request for Comments 2609, Network Working Group, Juni 1999. Category: Standards Track.

[Gutt01] E. Guttman. RFC 3059: Attribute List Extension for the Service Location Protocol. Request for Comments 3059, Network Working Group, Februar 2001. Category: Standards Track.

[HaPS02] Z. J. Haas, M. R. Pearlman und P. Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet Draft, MANET Working Group, Juli 2002.

[Hard02] T. Hardie. RFC 3258: Distributing Authoritative Name Servers via Shared Unicast Addresses. Request for Comments 3258, Network Working Group, April 2002. Category: Informational.

[Heer05] M. Heerwagen. Genauigkeits- und Verfügbarkeitsanalyse von Positionsbestimmungsverfahren. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Januar 2005.

[Henn06] J. Henniger. Simulation kontextsensitiver Routingprotokolle. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Oktober 2006.

[Hess05] A. Hesse. Untersuchung der Eignung von WLAN zur Übertragung multimedialer Daten. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Juni 2005.

[HHMN+01] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner und A. Schade. DEAPspace – Transient ad hoc networking of pervasive devices. *Computer Networks* Band 35, Issue 4, März 2001, S. 411–428.

[Hier07] G. R. Hiertz. Funk-Maschen: Standard für WLAN-Mesh-Netze erreicht Entwurfsstatus. *c't* (5), Februar 2007, S. 208–209.

[HiPa04] H. Hildebrandt und K. Payn. Bluetooth-Anwendungen. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2004.

[Hoff04] F. Hoffmann. Bluetooth-Anwendungen. Hauptseminar, Technische Universität Ilmenau, Ilmenau, April 2004.

[HoRa86] D. Hong und S. S. Rappaport. Traffic model and performance analysis for cellular mobile radio telephone systems with prioritized and nonprioritized handoff procedures. *IEEE Transaction on Vehicular Technology* 35(3), August 1986, S. 77–92. ISSN: 0018-9545.

[JaCl03] P. Jacquet und T. Clausen. RFC 3626: Optimized Link State Routing Protocol (OLSR). Request for Comments 3626, Network Working Group, Oktober 2003. Category: Experimental.

[JaWu05] I. Jawhar und J. Wu. QoS Support in TDMA-based Mobile Ad Hoc. *Journal of Computer Science and Technology* 20(6), November 2005, S. 797–810.

[JHMJ01] J. G. Jetcheva, Y.-C. Hu, D. A. Maltz und D. B. Johnson. A Simple Protocol for Multicast and Broadcast in Mobile Ad Hoc Networks. Internet Draft, IETF MANET Working Group, Juli 2001.

[JiLi99] M. Jiang und J. Li. Cluster Based Routing Protocol (CBRP). Internet Draft, Mobile Ad hoc Networking Working Group, August 1999.

[JMLV+01] P. Jaquet, P. Minet, A. Laouiti, L. Viennot, T. Clausen und C. Adjih. Multicast Optimized Link State Routing. Internet Draft, IETF MANET Working Group, November 2001.

[JoHM07] D. Johnson, Y. Hu und D. Maltz. RFC 4728: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. Request for Comments 4728, Network Working Group, Februar 2007. Category: Experimental.

[JoMH03] D. B. Johnson, D. A. Maltz und Y.-C. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet Draft, IETF MANET Working Group, April 2003.

[Jugl00] E. Jugl. *Mobilitätsmodellierung und Einflüsse auf Systemparameter von Mobilfunksystemen.* Dissertation, Technische Universität Ilmenau, Fakultät für Elektrotechnik und Informationstechnik, Ilmenau, Oktober 2000.

[KaTG03] V. Kawadia, Y. Thang und B. Gupta. System Services for Implementing Ad-Hoc Routing: Architecture, Implementation and Experiences. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, USA, Juni 2003. S. 99–112.

[KaWr00] D. Katabi und J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). In *ACM SIGCOMM*, September 2000, S. 3–15.

[KeGo01] J. Kempf und J. Goldschmidt. RFC 3082: Notification and Subscription for SLP. Request for Comments 3082, Network Working Group, März 2001. Category: Experimental.

[KeGu99] J. Kempf und E. Guttman. RFC 2614: An API for Service Location. Request for Comments 2614, Network Working Group, Juni 1999. Category: Informational.

[Klei06] T. Kleinz. Funk-Kommunen. *c't* (20), September 2006, S. 174–176.

[Klei07] L. Klein-Berndt. Kernel AODV. Technischer Bericht, National Institute of Standards and Technology (NIST), http://w3.antd.nist.gov/wctg/aodv_kernel/, Dezember 2007. Version 2.2.2.

[KoPe02] R. Koodli und C. E. Perkins. Service Discovery in On-Demand Ad Hoc Networks. Internet Draft, Seamoby Working Group, Oktober 2002.

[Koss06] A. Kossel. Internet: Deutschland holt auf. *c't* (26), Dezember 2006, S. 60.

[Kram05] M. Kramer. Realisierung einer Testumgebung für Ad-hoc-Netze. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, April 2005.

[KrDS07] P. Krasovsky, M. Debes und J. Seitz. Media-Push Technology in Multimedia Mobile Environment. In *4th IEEE Consumer Communication and Networking Conference (CCNC'07)*, Las Vegas, Nevada USA, Januar 2007. IEEE, S. 1182–1184. ISBN: 1-4244-0667-6.

[LeCa98] S.-B. Lee und A. T. Campbell. INSIGNIA: In-band signaling support for QOS in mobile ad hoc networks. In *5th International Workshop on Mobile Multimedia Communication (MoMuC)*, Berlin, Germany, Oktober 1998.

[LeCF06] J. Leguay, V. Conan und T. Friedman. QoS Routing in OLSR with Several Classes of Service. In *Proceedings IEEE PerCom Workshop on Pervasive Wireless Networking (PWN06)*, 2006.

[LeDS05] A. Lewandowska, M. Debes und J. Seitz. An Architecture for Context-Sensitive Telecommunication Applications. In J. Cordeiro, V. Pedrosa, B. Encarnacao und J. Filipe (Hrsg.), *WEBIST – 2005 First International Conference on Web Information Systems and Technologies*, Miami, USA, May 26 – 28 2005. Institute for Systems and Technologies of Information, Control and Communication (INSTICC), INSTICC PRESS, S. 40–47. ISBN 972-8865-20-1.

[LeRo97] Z. Lei und C. Rose. Probability criterion based location tracking approach for mobilitymanagement of personal communications systems. In *IEEE Global Telecommunication Conference (GLOBECOM)*, Band 2, Phoenix, AZ, USA, November 1997. S. 977–981. ISBN: 0-7803-4198-8.

[LeRo98] Z. Lei und C. Rose. Wireless Subscriber Mobility Management using Adaptive Individual Location Areas for PCS Systems. In *IEEE International Conference on Communications*, Band 3, Atlanta, GA, USA, Juni 1998. S. 1390–1394. ISBN: 0-7803-4788-9.

[Lewa07] A. Lewandowska. Architektur für die Verarbeitung von Kontextinformationen: Architekturkonzept. Fachpublikation – Forschungsbericht, Digitale Bibliothek Thüringen, http://www.db-thueringen.de/servlets/DocumentServlet?id=9077, Oktober 2007.

[LiLa05] L. Li und L. Lamont. A Lightweight Service Discovery Mechanism for Mobile Ad Hoc Pervasive Environment Using Cross-layer Design. In *Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops)*. IEEE Computer Society, 2005.

[LLPC+01] R. Leung, J. Liu, E. Poon, A. Chan und B. Li. MP-DSR: A QoS-Aware Multi-Path Dynamic Source Routing Protocol for Wireless Ad-Hoc Networks. In *Proceedings of 26th Annual IEEE Conference on Local Computer Networks (LCN 2001)*, 2001, S. 132–141.

[LSHG+00] S.-J. Lee, W. Su, J. Hsu, M. Gerla und R. Bagrodia. A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols. In *INFOCOM (2)*, Tel Aviv, Israel, März 2000. S. 565–574.

[LZHJ+06] M. J. Lee, J. Zheng, X. Hu, H. Juan, C. Zhu, Y. Liu, J. S. Yoon und T. N. Saadawi. A New Taxonomy of Routing Algorithms for Wireless Mobile Ad Hoc Networks: The Component Approach. *IEEE Communication Magazine* 44(11), November 2006, S. 116–123.

[Male08] Jaroslaw Malek. Trace graph – Network Simulator NS-2 trace file analyser. http://www.tracegraph.com/, Januar 2008.

[Marr06] A. Marr. Ad-hoc-Netze / heterogene Netze. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, April 2006.

[MIPv07] Internet Protocol Version 6 Multicast Addresses. http://www.iana.org/assignments/ipv6-multicast-addresses, November 2007.

[MIPv08] Internet Multicast Addresses. http://www.iana.org/assignments/multicast-addresses, Februar 2008.

[MuGLA96] S. Murthy und J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *Mobile Networks and Applications* 1(2), 1996, S. 183–197.

[NaNS98] T. Narten, E. Nordmark und W. Simpson. RFC 2461: Neighbor Discovery for IP Version 6 (IPv6). Request for Comments 2461, Network Working Group, Dezember 1998. Category: Standards Track.

[Nick07] D. Nicklas. Nexus–A Global, Active, and 3D Augmented Reality Model. In Dieter Fritsch (Hrsg.), *Photogrammetic Week '07*. Heidelberg: Herbert Wichmann Verlag, September 2007, S. 325–334. ISBN: 978-3-87907-452-5.

[Nies02] N. Niestroy. Grundlegende Merkmale kontextsensitiver Dienste und ihre Anwendung. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2002.

[Nord07] E. Nordström. AODV-UU (Ad-hoc On-demand Distance Vector Routing – For real world and simulation). Technischer Bericht, Uppsala University, Department of Information Technology, Uppsala, Schweden, Dezember 2007. http://core.it.uu.se/core/index.php/AODV-UU.

[Nowa06] S. Nowak. Ad-hoc-Netze / Multimediale Anwendungen. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, April 2006.

[NS208] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns/, Januar 2008.

[Obje04] Object Management Group, Inc. *Common Object Request Broker Architecture: Core Specification*, März 2004. Version 3.0.3.

[Ogie01] R. Ogier. TBRPF Multicast Routing. Online Publication (www.sri.com), Stanford Research Institute (SRI), Engineering & Systems Division, Menlo Park, CA, USA, November 2001.

[OgTL04] R. Ogier, F. Templin und M. Lewis. RFC 3684: Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). Request for Comments 3684, Network Working Group, Februar 2004. Category: Experimental.

[OLPC08] One Laptop per Child (OLPC). http://laptop.org, März 2008.

[OMNe08] OMNeT++ Discrete Event Simulation System. http://www.omnetpp.org/index.php, Januar 2008.

[OPNE08] OPNET. http://www.opnet.com/, Januar 2008.

[OSGi07] OSGi Alliance. *OSGi Service Platform Core Specification*, April 2007. Release 4, Version 4.1.

[owrt08] OpenWrt. http://openwrt.org, Januar 2008.

[PaCo01] V. Park und S. Corson. Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification. Internet Draft, IETF MANET Working Group, Juli 2001.

[PaMM93] C. Partridge, T. Mendez und W. Milliken. Host Anycasting Service. Request for Comments 1546, RFC 1546: Network Working Group, November 1993. Category: Informational.

[PaRM98] J. Pascoe, N. Ryan und D. Morse. Human-Computer-Giraffe Interaction – HCI in the field. In *Proceedings of the Workshop on Human Computer Interaction with Mobile Devices*, Glasgow, Scotland, 1998.

[Pasc01] R. Pascoe. Building Networks on the Fly. *IEEE Spectrum*, März 2001, S. 61–65.

[Pasc05] G. Paschold. Entwurf und Simulation eines kontextsensitiven Routingprotokolls. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, März 2005.

[PeBh94] C. E. Perkins und P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, S. 234–244.

[PeBR01] C. E. Perkins und E. M. Belding-Royer. Quality of Service for Ad hoc On-Demand Distance Vector Routing. Internet Draft, Mobile Ad hoc Networking Working Group, November 2001.

[PeBRD01] C. E. Perkins, E. M. Belding-Royer und S. R. Das. IP Flooding in Ad hoc Mobile Networks. Internet Draft, Mobile Ad Hoc Networking Group, November 2001.

[Pein04] K. Pein. Handover in heterogenen Netzen. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2004.

[PePC06] L. Pelusi, A. Passarella und M. Conti. Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. *IEEE Communication Networks* 44(11), November 2006, S. 134–141.

[PeRD00] C. E. Perkins, E. M. Royer und S. R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing for IP version 6. Internet Draft, Mobile Ad Hoc Networking Working Group, November 2000.

[Perk02] C. E. Perkins. RFC 3344: IP Mobility Support for IPv4. Request for Comments 3344, Network Working Group, August 2002. Category: Standards Track.

[Post81a] J. Postel. RFC 791: Internet Protocol. Request for Comments 791, Information Sciences Institute, University of Southern California, Marina del Rey, California, September 1981. DARPA Internet Program Protocol Specification.

[Post81b] J. Postel. RFC 792: Internet Control Message Protocol. Request for Comments 792, Network Working Group, September 1981. DARPA Internet Program Protocol Specification.

[Pott03] C. Potthoff. Barrierefreier Campus: Lokalisierung; Lokalisierung im freien Feld. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Dezember 2003.

[Pott04] C. Potthoff. Erhöhung der Positioniergenauigkeit von GPS-Empfängern durch Nutzung von Assistenzinformationen. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, November 2004.

[Psch06] A. Pschichholz. Ad-hoc-Netze / kontextsensitive Anwendungen. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, April 2006.

[ReLi95] Y. Rekhter und T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4). Request for Comments 1771, Network Working Group, März 1995. Category: Standards Track.

[Renh05] K. Renhak. WiMAX – der neue IEEE-Standard für den drahtlosen Breitbandzugang. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Januar 2005.

[Renh06] K. Renhak. Router mit Kontextwissen. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, August 2006.

[Renh07a] K. Renhak. Entwicklung einer Clientsoftware zur Unterstützung des kontextsensitiven Routings. Zwischenverteidigung – Diplomarbeit, Technische Universität Ilmenau, Ilmenau, Dezember 2007.

[Renh07b] K. Renhak. Service-Discovery-Protokolle. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, April 2007.

[Renh08] K. Renhak. Entwicklung einer Clientsoftware zur Unterstützung des kontextsensitiven Routings. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, März 2008.

[RoPe99] E. M. Royer und C. E. Perkins. Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol. In *Proceedings of the Fifth Annual ACM/IEEE Conference on Mobile Computing and Networking*, 1999, S. 207–218.

[RyPM97] N. Ryan, J. Pascoe und D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney (Hrsg.), *Computer Applications in Archaeology*, 1997.

[Schi03] J. Schiller. *Mobilkommunikation*. Pearson Studium, München. 2. Auflage, 2003.

[Schm05] E. Schmidt. Adressierung für kontextsensitives Routing. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, März 2005.

[SDHT07] J. Seitz, M. Debes, M. Heubach und R. Tosse. *Digitale Sprach- und Datenkommunikation: Netze, Protokolle, Vermittlung*. Fachbuchverlag Leipzig im Carl Hanser Verlag, München, Wien. ISBN: 978-3-446-22979-2, 1. Auflage, 2007.

[Seni99] D. Senie. RFC 2644: Changing the Default for Directed Broadcast in Routers. Request for Comments 2644, Network Working Group, August 1999. Category: Best Current Practice.

[SFB06] SFB 627: Nexus – Umgebungsmodelle für Mobile Kontextbezogene Systeme. http://www.nexus.uni-stuttgart.de/index.html, Dezember 2006.

[SFB07] SFB 627: Teilprojekt – Kontextbezogene Kommunikation. http://www.nexus.uni-stuttgart.de/de/forschung/teilprojekte/tpa2/tpa2.html, November 2007.

[SiDB98] R. Sivakumar, B. Das und V. Bharghavan. Spine routing in ad hoc networks. *ACM/Baltzer Cluster Computing Journal* 1(2), 1998, S. 237–248.

[SiSB98] R. Sivakumar, P. Sinha und V. Bharghavan. Core Extraction Distributed Ad hoc Routing (CEDAR) Specification. Internet Draft, IETF MANET Working Group, Oktober 1998.

[SiSB99] P. Sinha, R. Sivakumar und V. Bharghavan. MCEDAR: Multicast core extraction distributed ad-hoc routing. In *Proceedings of the Wireless Communications and Networking Conference*, 1999.

[SKATL+99] A. Schmidt, A. Takalumoma K. Aidoo, U. Tuomela, K. Van Laerhoven und W. Van de Velde. Advanced Interaction in Context. In *1st International Symposium on Handheld and Ubiquitous Computing (HUC99)*, Band 1707 der *Lecture notes in computer science*. Springer, 1999, S. 89–101.

[SrHo01] P. Srisuresh und M. Holdrege. RFC 3027: Protocol Complications with the IP Network Address Translator. Request for Comments 3027, Network Working Group, Januar 2001. Category: Informational.

[Stoj02] I. Stojmenovic. Position-Based Routing in Ad Hoc Networks. *IEEE Communications* 40(7), Juli 2002, S. 128–134.

[Sun 01] Sun Microsystems, Inc., Palo Alto, California, USA. *Jini Architecture Specification*, Version 1.2. Auflage, Dezember 2001.

[Teng06] R. Teng. PBR: A Region-Based Routing Protocol for Mobile Nodes in Hybrid Ad Hoc Networks. *IEEE Communications Magazine* 44(11), November 2006, S. 124–132.

[ThGM88] R. Thomas, H. Gilbert und G. Mazziotto. Influence of the Movement of the Mobile Station on the Performance of a Radio Cellular Network. In *Proceedings of the 3rd Nordic Seminar*, Copenhagen, September 1988.

[ToGM04] D. A. Torres und J. A. Garcia-Macias. Service Discovery in Mobile Ad-hoc Networks by Extending the AODV Protocol. In *Second Mobile Computing Workshop (ENC'04)*, ISBN: 970-692-170-2, Colima, Mexico, September 2004.

[Toh99] C.-K. Toh. Long-lived Ad Hoc Routing based on the Concept of Associativity. Internet Draft, IETF MANET Working Group, März 1999.

[Toh01] C.-K. Toh. Maximum Battery Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks. *IEEE Communication Magazine* 39(6), Juni 2001, S. 138–147.

[Toh02] C.-K. Toh. *Ad Hoc Mobile Wireless Networks – Protocols and Systems.* Prentice Hall PTR, Upper Saddle River, New Jersey. ISBN: 0-13-007917-4, 2002.

[TyNo08] ICMP Type Numbers. http://www.iana.org/assignments/icmp-parameters, Februar 2008.

[UoBJ07] UoB JAdhoc – AODV Implementation in Java. http://www.aodv.org/, Dezember 2007. Version: 0.21, Universität Bremen.

[UoBW07] UoBWinAODV – Windows AODV Protocol Handler. http://www.aodv.org/, Dezember 2007. Version: 0.15, Universität Bremen.

[UPnP06] UPnP Forum. *UPnP Device Architecture 1.0*, Document Version 1.0.1. Auflage, Juli 2006. http://www.upnp.org/specs/arch/UPnP-DeviceArchitecture-v1.0-20060720.pdf.

[Vosw04] F. Voswinkel. Barrierefreier Campus: Teilprojekt Benutzerprofile. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Januar 2004.

[W11n07] IEEE 802.11n. http://de.wikipedia.org/wiki/802.11n, Mai 2007.

[Wang02] Z. Wang. *An Agent-Based Integrated Service Platform for Wireless and Mobile Environments.* Dissertation, Universtät Karlsruhe (TH), Fakultät für Informatik, Karlsruhe, November 2002.

[WeCh04] S. Weber und L. Cheng. A survey of Anycast in IPv6 Networks. *IEEE Communications Magazine* 42(1), Januar 2004, S. 127–132.

[Weis91] M. Weiser. The Computer of the 21st Century. *Scientific American* 265(3), September 1991, S. 66–75.

[Wenz06] M. Wenzel. Evaluierung eines modularen Routers und Implementierung eines Ad-hoc-Routingprotokolls. Studienarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Dezember 2006.

[Wenz07a] M. Wenzel. Konzeption und Umsetzung eines Demonstrators zur Verifikation kontextsensitiver Routingprotokolle. Diplomarbeit, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, September 2007.

[Wenz07b] M. Wenzel. UWB – Ultrawideband-Kommunikation. Hauptseminar, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Januar 2007.

[Wind07] C. Windeck. Schnellverbinder: Das leisten Chipsätze auf PC-Mainboards. *c't* (3), Januar 2007, S. 206–213.

[Wint03] S. Winter. Barrierefreier Campus: Lokalisierung in Gebäuden. Medienprojekt, Technische Universität Ilmenau, Fachgebiet Kommunikationsnetze, Ilmenau, Dezember 2003.

[wnte03] Wireless Network Topology Emulator (WNTE). http.sourceforge.net, November 2003. Release 2.0-scripts.

[wrt008] DD-WRT. http://www.dd-wrt.com, Januar 2008.

[WSWZ04] J. Wu, O. Stanze, K. Weniger und M. Zitterbart. Prototype Implementation of Anycast-based Service Discovery for Mobile Ad Hoc Networks, 18. DFN-Arbeitstagung über Kommunikationsnetze, Juni 2004.

[ZhJa04] X. Zhang und L. Jacob. MZRP: An Extension of the Zone Routing Protocol for Multicasting in MANETs. *Journal of Information Science and Engineering* 20(3), Mai 2004, S. 535–551.

[ZhKu04] Y. Zhu und T. Kunz. MAODV Implementation for NS-2.26. Technical report sce-04-01, Carleton University, Systems and Computer Engineering, Ottawa, Canada, Januar 2004.

[ZoDa97] M. M. Zonoozi und P. Dassanayake. A Novel Modelling Technique for Tracing Mobile Users in a Cellular Mobile Communication System. *Wireless Personal Communication* 4(2), März 1997, S. 185–205. ISSN: 0929-6212.